ELSEVIER

Science of Computer Programming 23 (1994) 151-195

Science of Computer Programming

Model checking using net unfoldings [1]

Javier Esparza [2]

Institut für Informatik, Technische Universität München, Arcisstr. 21, 80290 München, Germany

Abstract

McMillan (1992) 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 new algorithm is shown to be polynomial in the size of the net for 1-safe conflict-free Petri nets, while the algorithms of the literature require exponential time.

1. Introduction

In the early eighties, Quielle and Sifakis [33], Clarke et al. [11] and others begun a new approach for the verification of finite-state concurrent systems. This approach is known as "model-checking", because the idea is to express a desirable property in some logic, view the system as a structure on which to interpret the logic by means of some formal semantics, and check whether this structure is a model of the formula. In a nutshell, "system enjoys property" is formalised as "system's semantics is model of formula".

Model checking was intensively studied in the next years. It proved to be simpler and more efficient than former approaches based on theorem proving, in which both the system and the formula are expressed in the logic, and "system enjoys property" is formalised as "system's formula implies property's formula". It was applied to the verification of communication protocols, asynchronous circuit designs and mutual exclusion algorithms, among others (see, for instance [8,40]). Model checkers were proposed

[1] A shortened version of this paper appeared in: M.C. Gaudel and J.P. Jouannaud, eds., *Proceedings of TAPSOFT '93*, Lecture Notes in Computer Science, Vol. 668 (Springer, Berlin, 1993) 613-628. This work was partially supported by ESPRIT WG 6067 CALIBAN.

[2] This work was done while the author was at the University of Hildesheim and at the University of Edinburgh. E-mail: esparza@informatik.tu-muenchen.de.

SSDI 0167-6423(94)00019-0

for a rich variety of logics (linear and branching time, state and event based, μ-calculi, etc.).

The main problem of finite-state model checking is the so called state explosion: the number of states of the system may grow exponentially in its size, which complicates or prevents the verification of large systems. Many different solutions have been proposed to cope with this problem. In [12], properties of networks with many identical processes are parametrically verified. In [1,24], methods for compositional verification — in which the verification of a property of a system is reduced to the verification of properties of its components — are described. Binary Decision Diagrams have been used in [9] to intelligently encode state spaces which exhibit some kind of regularity. Other researchers have observed that the arbitrary interleaving of concurrent actions — that is, the identification of "a is concurrent with b" with "a can occur before b and b can occur before a" — greatly contributes to the state explosion problem. This paper is also based on this observation; it proposes a new model checking algorithm based on net unfoldings, a well studied partial order semantics of Petri nets.

In order to motivate this new approach, it is convenient to briefly overview the verification algorithms based on partial orders described in the literature. In a series of papers, Valmari has developed the stubborn set method. Originally, the method constructed a reduced state space in which some interleavings were removed while preserving some properties, essentially the existence of deadlocks [38]. Later on, Valmari et al. extended the technique by considering reductions that preserve different equivalences (see, for instance, [39]). More than a partial order method, the stubborn set method is the application of a partial order notion to the interleaving method in order to cleverly obtain reduced state spaces.

The method of Godefroid et al. (see, for instance, [21,22]) uses Mazurkiewicz's traces as semantic model. It constructs a trace automaton, which can also be considered as a sort of reduced state space in which at least one interleaving of each trace is present. Very general logics can be checked due to the use of "on the fly" verification: the system, represented by a product of automata or a 1-safe Petri net, is composed with an automaton corresponding to the negation of the property to be verified. Then, an equivalent trace automaton is computed and checked for nonemptyness. "On the fly" verification has the advantage that it may reduce the size of the final trace automaton in favourable cases, and the disadvantage that the verification has to start anew for each new property. The method makes light use of trace theory, and concentrates more on algorithmical aspects.

Probst and Li [32] have developed a method based on an *ad hoc* partial order semantics called "behaviour machines". Properties are not specified within a logic, but by semantical methods. This method is difficult to compare with the others due to the particular way in which both systems are properties are modeled, and we will not consider it any further.

Finally, McMillan proposes in [29] a method for deadlock detection, which, as we shall see, is one of the direct sources of inspiration of this paper. The method is based on net unfoldings [6,16,31]. Net unfoldings can be seen as concurrent versions of the

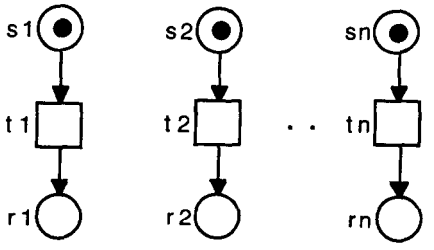

Fig. 1. A simple class of concurrent systems.

usual notion of unfolding of a loop. They have the problem of being infinite whenever the system may exhibit an infinite computation (this is usually the case in reactive systems), which complicates their use for verification. McMillan proposes an elegant algorithm for the computation of a finite initial part of the unfolding in which every reachable marking is represented.

This variety of proposals has been successfully applied to a number of examples. However, there are still open points which have not been addressed in a satisfactory way. None of the proposals matches the clear and well defined structure of interleaving model checkers, in which a logic is given and a state space computed which can be repeatedly used to check every formula of the logic; it would be interesting to develop a partial order model checker following this pattern. Also, partial orders are used to obtain compact semantics, but the more sophisticated results on partial orders are not used to improve the efficiency of the search algorithms. Finally, there exist properties for which these proposals (when applicable) have a too large computational cost.

We consider this last point in more detail in an example. Consider the family of systems composed by a number n of independent subsystems which can just execute an action. A Petri net model of this family is shown in Fig. 1. The reachability graph of the systems of this class contains 2^n markings, where n is the number of transitions. It should be clear that the only reason of this combinatorial explosion is the representation of concurrency by interleaving. Let us examine the methods of Valmari, Godefroid and McMillan on the problem of deciding if a given marking or submarking of one of these systems is reachable (a submarking is a partially specified marking). Valmari's method cannot be directly applied to this problem; if we remove a reachable marking M, then we can no longer check whether M is reachable by traversing the reduced state space. Therefore, no state space reduction can be applied. A blind application of Godefroid's method can take exponential time. The reason is that "on the fly" verification modifies the system according to the property that has to be checked. In this case, a new transition is added, having as input places the set of places that have to get a token at the specified marking.[3] After this addition, the system is no longer composed of n independent subsystems, and, in the worst case, the whole state space of the system may have to be generated before the desired marking is found. By means of the same

[3] Some other places and transitions have to be added as well, but this is not relevant for our discussion.

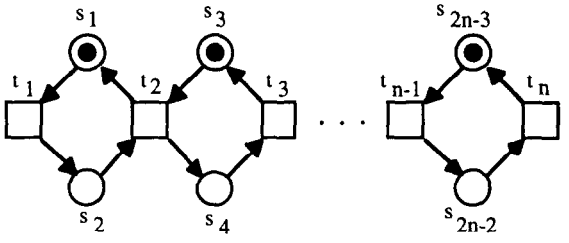

Fig. 2. A simplified model of a concurrent buffer of capacity $n-1$.

addition, McMillan's method can also be used to decide the reachability of a marking, but it faces the same problem.

Of course, all these methods can solve our problem in polynomial time, if we just modify them so that they can recognize independent subsystems, but this is an *ad hoc* solution which, for instance, will not be applicable to the next example: the family of systems shown in Fig. 2. These systems model concurrent buffers of different capacities. The buffer consists of a certain number of cells, but the cells are no longer independent. A token in one odd place represents that the corresponding cell is empty, and a token in an even place that the corresponding place is full. The items enter the buffer via the occurrence of t_1 and leave it via the occurrence of t_n. It is not difficult to see that the number of reachable markings grows exponentially in the number of cells. We consider again the problem of the reachability of a marking (for the marking that puts a token in all places with an even index, this is the problem of deciding if all cells can simultaneously be full). Now, none of the algorithms discussed above can give an answer in polynomial time, and there seems to be no simple way to modify them so that the state explosion is avoided. However, the problem is polynomial in the size of the system: a well known result of net theory [14,20] states that the marking is reachable if and only if a certain set of linear equations, whose size is roughly equal to the size of the system, has a solution. So we are lead to the conclusion that, for certain properties such as the reachability of a state, partial order model checking has not yet been able to palliate the state explosion problem.

In this paper, we show how a combination of McMillan's idea and a deeper use of the theory of net unfoldings can provide solutions to the open points that have been mentioned. We construct a model checker in which the state space is replaced by McMillan's unfolding; this unfolding can be stored and used to check all the properties of a temporal logic with a possibility operator able to express properties as reachability, mutual exclusion or the liveness of a transition.

To support the interest of the model checker, we first prove that it is linear in the size of McMillan's unfolding for the class of persistent nets; this unfolding can be much smaller than the state space. Furthermore, we show that the model checker is polynomial in the size of the net for 1-safe conflict-free nets. The class of conflict-free nets have been very thoroughly studied in the literature [17,25–27,41], and contains the two examples we have discussed in this introduction. In [17,41] it has been proved that two particular problems on 1-safe conflict-free nets can be solved in polynomial time in the

size of the net. Our result subsumes these two as special cases, and extends the range of polynomial properties to all those expressible in a logic.

2. Nets and branching processes

Many different classes of Petri nets have been described in the literature. We first specify the class we consider as system models. Then, we introduce Engelfriet's branching processes [16]. Finally, we briefly recall the notions of configuration and cut.

2.1. 1-Safe Petri net systems

We choose a basic model of Petri nets in which places can contain at most one unstructured token, and have therefore a boolean character (marked/unmarked). In [3], a survey on different classes of Petri nets, they are called 1-safe net systems. A triple (S,T,F) is a *net* if $S \cap T = \emptyset$ and $F \subseteq (S \times T) \cup (T \times S)$. The elements of S are called *places*, and the elements of T *transitions*. Places and transitions are generically called *nodes*. We identify F with its characteristic function on the set $(S \times T) \cup (T \times S)$. The *preset* of a node x, denoted by ${}^\bullet x$, is the set $\{y \in S \cup T \mid F(y,x) = 1\}$. The *postset* of x, denoted by $x^\bullet$, is the set $\{y \in S \cup T \mid F(x,y) = 1\}$.

A *marking* of a net (S,T,F) is a mapping $S \to \mathbb{N}$. A fourtuple $\Sigma = (S,T,F,M_0)$ is a *net system* if (S,T,F) is a net and M_0 is a marking of (S,T,F) (called the *initial marking* of Σ). A marking M *enables* a transition t if $\forall s \in S: F(s,t) \leqslant M(s)$. If t is enabled at M, then it can *occur*, and its occurrence leads to a new marking M' (denoted $M \xrightarrow{t} M'$), defined by $M'(s) = M(s) - F(s,t) + F(t,s)$ for every place s. A sequence of transitions $\sigma = t_1 t_2 \ldots t_n$ is an *occurrence sequence* if there exist markings M_1, M_2, ..., M_n such that

$$M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \cdots \xrightarrow{t_{n-1}} M_{n-1} \xrightarrow{t_n} M_n$$

M_n is the marking reached by the occurrence of σ, also denoted by $M_0 \xrightarrow{\sigma} M_n$. M is a *reachable marking* if there exists an occurrence sequence σ such that $M_0 \xrightarrow{\sigma} M$.

A marking M of a net is *1-safe* if $M(s) \leqslant 1$ for every place s. We identify 1-safe markings with the set of places s such that $M(s) = 1$. A net system Σ is 1-safe if all its reachable markings are 1-safe.

This definition of 1-safe systems corresponds to the one given in [3]. However, it is convenient to exclude from our considerations some 1-safe systems, for instance those containing isolated nodes, i.e., places or transitions which are not connected to any other node. In the sequel, we call 1-safe systems those satisfying the following four properties:

- The number of places and transitions is finite.
- Every place has a nonempty preset *or* a nonempty postset.
- Every transition of T has a nonempty preset *and* a nonempty postset.
- The system is 1-safe in the sense of the definition above.

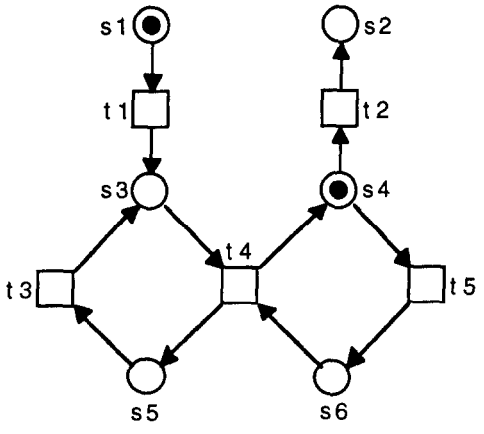

Fig. 3. A 1-safe system.

Fig. 3 shows a 1-safe system in this restricted sense. According to our convention, the initial marking is $\{s_1, s_4\}$. Elementary Net Systems and Condition/Event Systems [3] are other net models in which places contain at most one unstructured token. Our results are valid for all these models.

2.2. *Branching processes*

Branching processes are unfoldings of net systems containing information about both concurrency and conflicts. Engelfriet developed in [16] a theory of branching processes similar to the theory of processes (see [6,36]). Branching processes are based on the notion of net unfoldings introduced by Nielsen et al. in [31].

In this section, we quickly review the main definitions and results of [16]. Notice that these definitions are given for net systems which are not necessarily 1-safe.

Let (S,T,F) be a net and let $x_1, x_2 \in S \cup T$. The nodes 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 (t_1, x_1), (t_2, x_2) belong to the reflexive and transitive closure of F. In other words, x_1 and x_2 are in conflict if there exist two paths leading to x_1 and x_2 which start at the same place and immediately diverge (although later on they can converge again). For $x \in S \cup T$, x is in *self-conflict* if $x \# x$.

An *occurrence net* is a net $N = (B, E, F)$ such that:

- for every $b \in B$, $|{}^\bullet b| \leqslant 1$,
- the (irreflexive) transitive closure of F is acyclic,
- N is finitely preceded, i.e., for every $x \in B \cup E$, the set of elements $y \in B \cup E$ such that (y, x) belongs to the transitive closure of F is finite, and
- no event $e \in E$ is in self-conflict.

The elements of B and E are called *conditions* and *events*, respectively. $Min(N)$ denotes the set of minimal elements of $B \cup E$ with respect to the transitive closure of F. A *causal*

net is an occurrence net which also satisfies $|b^\bullet| \leqslant 1$ for every condition b.[4] Observe that no two nodes of a causal net are in conflict.

We now define homomorphisms of nets.[5] Let $N_1 = (S_1, T_1, F_1)$ and $N_2 = (S_2, T_2, F_2)$ be two nets. A *homomorphism* from N_1 to N_2 is a mapping $h: S_1 \cup T_1 \to S_2 \cup T_2$ such that:

- $h(S_1) \subseteq S_2$ and $h(T_1) \subseteq T_2$, and
- for every $t \in T_1$, the restriction of h to ${}^\bullet t$ is a bijection between ${}^\bullet t$ (in N_1) and ${}^\bullet h(t)$ (in N_2), and similarly for $t^\bullet$ and $h(t)^\bullet$.

In other words, a homomorphism is a mapping that preserves the nature of nodes and the environment of events.

A branching process of a net system $\Sigma = (N, M_0)$ is a pair $\beta = (N', p)$ where $N' = (B, E, F)$ is an occurrence net, and p is a homomorphism from N' to N such that

(i) The restriction of p to $Min(N')$ is a bijection between $Min(N')$ and M_0,

(ii) for every $e_1, e_2 \in E$, if ${}^\bullet e_1 = {}^\bullet e_2$ and $p(e_1) = p(e_2)$ then $e_1 = e_2$.

If N' is a causal net, then β is a *process* of Σ.

Fig. 4 shows a branching process of the 1-safe system of Fig. 3, which will be used throughout this paper to illustrate definitions and algorithms. The name of a condition is written inside its corresponding circle. Beside the circle is the name of the place assigned to the condition by the homomorphism of the branching process. The same graphical convention is used for events. We say that the names of the places and transitions *label* the conditions and events. Accordingly, a homomorphism is also called a *labelling*). There exists a natural relation between branching processes of a system, called the approximation relation, which corresponds to the idea "being an initial part of". For instance, the branching process of Fig. 4 having $\{b_1, \ldots, b_5\}$ as conditions and $\{e_1, \ldots, e_3\}$ as events is an initial part of the whole branching process shown in the figure. We now formalise this relation.

Let $\beta_1 = (N_1, p_1)$ and $\beta_2 = (N_2, p_2)$ be two branching processes of a net system. A *homomorphism* from β_1 to β_2 is a homomorphism p from N_1 to N_2 such that

- the restriction of p to $Min(N_1)$ is a bijection between $Min(N_1)$ and $Min(N_2)$, and
- $p_2 \circ p = p_1$, where $\circ$ denotes function composition.

β_1 and β_2 are *isomorphic* if there exists a bijective homomorphism from β_1 to β_2. We say that β_1 *approximates* β_2 if there exists an injective homomorphism from β_1 to β_2. In particular, if (N, p) and (N', p') are branching processes such that N' is a subnet of N and p' is the restriction of p to N', then (N', p') approximates (N, p); the required injective homomorphism is just the identity.

It is shown in [16] that the set of isomorphism classes of branching processes of a net system is a complete lattice with respect to the approximation relation. In particular, a system has a unique maximal branching process up to isomorphism. The maximal branching process (up to isomorphism) of the 1-safe system of Fig. 3 is infinite.

[4] Causal nets are called occurrence nets in [6]. We follow here the terminology of [16].

[5] In [16], homomorphisms are defined between net systems, instead of between nets, but this is only a small technical difference without any severe consequence.

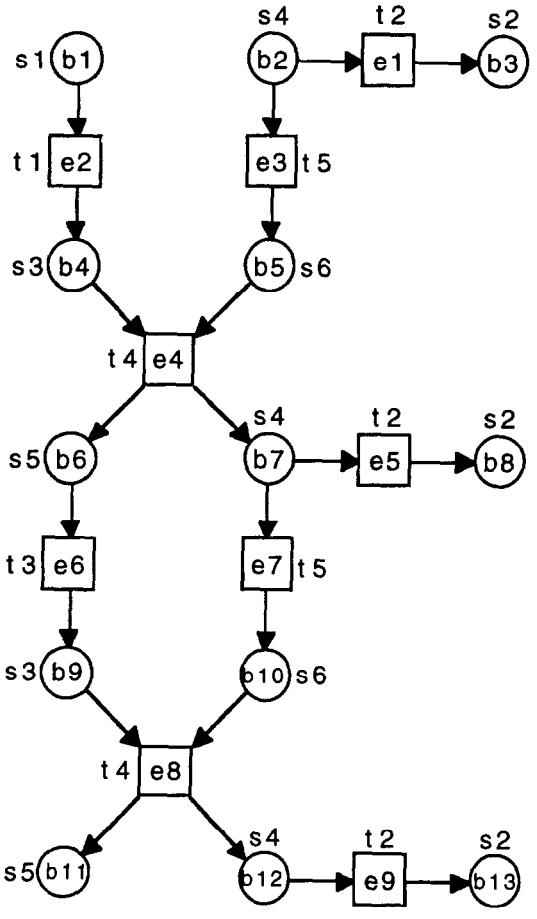

Fig. 4. A branching process of the 1-safe system of Fig. 3.

Loosely speaking, it consists of a periodic repetition of the part of the branching process of Fig. 4 obtained by horizontally "cutting" the net just below the event e_4.

2.3. *Configurations and cuts*

In an occurrence net (B, E, F), the transitive closure of the flow relation F is acyclic, and therefore a partial order. We call it the *causal relation*, and denote it by $\prec$. The symbol $\preceq$ denotes the reflexive and transitive closure of F. We generalise the causal relation to sets of nodes in the following way: given $x \in B \cup E$ and $X \subseteq B \cup E$, we say $x \prec X$ ($x \succ X$) if there exists $y \in X$ such that $x \prec y$ ($x \succ y$).

A *configuration* C of an occurrence net is a set of events satisfying the following two conditions:

- $e \in C \Rightarrow \forall e' \preceq e: e' \in C$ (C is causally closed).
- $\forall e, e' \in C: \neg(e \# e')$ (C is conflict-free).

A finite configuration can be seen as the set of events that have occurred up to a certain moment in a run of the system. More formally, if C is a configuration of the maximal branching process and $e_1\ e_2 \ldots e_n$ is a linearisation of the events of C (a total order on C which is compatible with the causal relation), then the corresponding sequence of labels $t_1\ t_2 \ldots t_n$ is an occurrence sequence. The converse also holds (see [6]).

The set of configurations of an occurrence net is partially ordered with respect to set inclusion, and closed under intersection. In the case of causal nets, it is also closed under union, and therefore a complete lattice. For arbitrary occurrence nets this latter property does not hold. We only have that, given a set of configurations $\mathcal{C}$, if the union of every two configurations of $\mathcal{C}$ is a configuration, then the union of all the elements of $\mathcal{C}$ is a configuration.

A set B' of conditions of an occurrence net is a *co-set* if

$$\forall b, b' \in B' : \neg(b \prec b') \wedge \neg(b' \prec b) \wedge \neg(b \# b')$$

A maximal co-set B' with respect to set inclusion is called a *cut*. Every co-set can be extended to a cut.

Intuitively, cuts correspond to the markings reached after the execution of the events contained in a finite configuration. Formally, we have that a set of places M is a reachable marking of a 1-safe system Σ iff there exists a cut c of the maximal branching process (N, p) of Σ such that $p(c) = M$ and $|p(c)| = |M|$. An easy consequence of this result is that any two conditions of a process carrying the same label are causally ordered.

Finite configurations and cuts are tightly related via the following mapping.

Definition 2.1 (*The mapping Cut*). Let C be a finite configuration of a branching process β with underlying net N. We associate to C a set of conditions $Cut(C)$ in the following way:

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

For the configuration $\{e_2\}$ of Fig. 4, we have

$$Cut(\{e_2\}) = (\{b_1, b_2\} \cup \{b_4\}) \setminus \{b_1\} = \{b_2, b_4\}$$

Cut is a bijective mapping from the set of finite configurations of an occurrence net onto its set of cuts. Therefore, it induces a relation $\sqsubseteq$ on the set of cuts corresponding to set inclusion in the set of finite configurations, and two operators $\sqcup$ and $\sqcap$ corresponding to set union and set intersection. The following characterisations are easy to prove. For every two cuts c_1, c_2:

$$c_1 \sqsubseteq c_2 \text{ iff } \forall b_1 \in c_1\ \exists b_2 \in c_2 : b_1 \preceq b_2 \text{ (or } \exists b_1 \in c_1\ \forall b_2 \in c_2 : b_1 \preceq b_2)$$

$$c_1 \sqcup c_2 = (c_1 \cup c_2) \setminus (\{b_1 \in c_1 \mid b_1 \prec c_2\} \cup \{b_2 \in c_2 \mid b_2 \prec c_1\})$$

$$c_1 \sqcap c_2 = (c_1 \cup c_2) \setminus (\{b_1 \in c_1 \mid b_1 \succ c_2 \vee b_1 \# c_2\} \cup \{b_2 \in c_2 \mid b_2 \succ c_1 \vee b_2 \# c_1\})$$

where $b \# c$ if there exists $b' \in c$ such that $b \# b'$.

The cuts of an occurrence net are partially order with respect to $\sqsubseteq$, and closed under $\sqcap$. In the case of causal nets, they are also closed under $\sqcup$, and therefore the set of configurations is a lattice (not necessarily complete).

Given a finite configuration C of a branching process $\beta = (N, p)$, the set of places $p(Cut(C))$ is a reachable marking. Since this composition of the mappings p and Cut will be frequently used, we give it a name.

Notation 2.2 (*The mapping Mark*). Let $\beta = (N, p)$ be a branching process of a 1-safe system Σ, and let C be a finite configuration of β. The reachable marking $p(Cut(C))$ is denoted by $Mark(C)$.

We use cuts in order to define the satisfaction relations of the logics we shall consider. However, we develop the theory in terms of configurations, mainly because we can use standard set theory to handle them (instead of special operators such as $\sqcup$ and $\sqcap$).

3. A model checker for a branching time logic

We define a simple branching time logic for 1-safe systems (i.e., a temporal logic in which every time instant may have several different successors which correspond to different possible futures) and design a model checker for it based on branching processes.

In the first section, we define the logic and briefly discuss its expressive power; also, we give some results about the complexity of the model checking problem, which impose limits upon the efficiency that can be expected from our model checker. The second section introduces the finite approximation of the maximal branching process on which the model checker is based. Sections 3–5 reduce the model checking problem to two simpler problems, and Section 6 provides algorithms to solve them. In the final section we give a global description of the model checker.

3.1. A branching time logic for 1-safe net systems

In this section, we define the syntax and semantics of a modal logic L tailored for 1-safe systems. We fix for the rest of the section a 1-safe system $\Sigma = (S, T, F, M_0)$. Σ has a unique maximal branching process up to isomorphism. We fix a representative of the isomorphism class, denoted by

$$\beta_m = (N_m, p_m) \qquad N_m = (B_m, E_m, F_m)$$

where the subscript m stands for "maximal".

The logic L extends propositional logic with a possibility operator. Its atomic sentences are places of a net.

Definition 3.1 (*Syntax of L*). The formulas of the logic L over Σ are those generated by the following grammar:

$\phi ::=$	**true**	(Truth)
	s where $s \in S$	(Place assertion)
	$\neg\phi$	(Negation)
	$\phi_1 \wedge \phi_2$	(Conjunction)
	$\Diamond\phi$	(Possibly ϕ).

The other boolean connectives are derived as usual, as well as the "box" operator $\Box = \neg\Diamond\neg$ (read $\Box\phi$ as "always ϕ").

Modal logics are interpreted over a set of worlds on which a binary relation is defined. In our case, the worlds are the finite configurations of the maximal branching process β_m, and the binary relation is set inclusion (see [23,34] for another logic interpreted on net unfoldings).

Definition 3.2 (*Semantics of L*). Let C be a finite configuration of β_m. We define inductively when C satisfies a formula ϕ over Σ.

$C \models s$	if $s \in Mark(C)$.
$C \models \neg\phi$	if not $C \models \phi$.
$C \models \phi_1 \wedge \phi_2$	if $C \models \phi_1$ and $C \models \phi_2$.
$C \models \Diamond\phi$	if $C' \models \phi$ for some configuration $C' \supseteq C$.

We say that Σ satisfies ϕ, also denoted by $\Sigma \models \phi$, if $\emptyset \models \phi$.

Informally 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.

The logic L can express properties such as:

- Reachability of a marking. For instance, the net system of Fig. 3 satisfies the formula $\Diamond(\neg s_1 \wedge s_2 \wedge \neg s_3 \wedge s_4 \wedge \neg s_5 \wedge \neg s_6)$ iff the marking $\{s_2, s_4\}$ is reachable.
- Mutual exclusion between places. The net system satisfies $\Box(\neg s_3 \vee \neg s_5)$ iff no marking puts tokens simultaneously on s_3 and s_5.
- Concurrency of transitions. The net system satisfies $\Diamond(s_4 \wedge s_5)$ iff the transitions t_3 and t_5 are concurrently enabled at some reachable marking.
- Liveness of a transition. The net system satisfies $\Box\Diamond s_4$ iff the transition t_5 is live.
- Cyclicity. The initial marking of the system can always be reached again iff the net system satisfies $\Box\Diamond(s_1 \wedge \neg s_2 \wedge \neg s_3 \wedge s_4 \wedge \neg s_5 \wedge \neg s_6)$.

These properties are among the most studied in net theory. Notice, however, that L cannot express progress properties.

The logic L has the same expressive power as its interleaving version interpreted on the reachable markings of the reachability graph. In this interpretation, we define

$M \models' s$ iff $s \in M$

$M \models' \Diamond\phi$ iff there exists M' reachable from M such that $M' \models' \phi$

The semantics of the boolean connectives is defined as before, and now Σ satisfies ϕ if $M_0 \models' \phi$. It follows from the definitions and theorems of Section 2 that $\Sigma \models \phi$ if and only if $\Sigma \models' \phi$.

This interleaving logic is equivalent to the fragment of the logic *UB* [2] containing only the operator *EF* and its dual *AG*.

The model checking problem for L is defined as the problem of deciding for a 1-safe system Σ and a formula ϕ whether $\Sigma \models \phi$ or not.

Formulas without $\Diamond$-symbols can be checked using the definition of $\models$ directly, because in order to decide if they hold it suffices to examine the initial marking of the system (for instance, $\Sigma \models s_1 \wedge \neg s_2$ if the initial marking of Σ puts one mark in s_1 and no mark in s_2). Therefore, the model checking problem reduces to the subproblem in which the formula is of the form $\Diamond\phi$.

We assign to a formula the set of finite configurations that satisfy it.

Notation 3.3 (*Denotation of a formula*). $Sat(\phi)$ denotes the set of finite configurations of β_m that satisfy ϕ.

Using this notion, we can denotationally formulate the model checking problem as the problem of deciding, given a formula $\Diamond\phi$, if $Sat(\phi)$ is empty.

Proposition 3.4 (Denotational formulation of the model checking problem). *Let ϕ be a formula over Σ. $\Sigma \models \Diamond\phi$ iff $Sat(\phi) \neq \emptyset$.*

Proof. By the definition of $\models$, $\Sigma \models \Diamond\phi$ iff some configuration $C \supseteq \emptyset$ satisfies ϕ. This is the case iff $Sat(\phi) \neq \emptyset$. □

The size of an instance of the model checking problem is given by the sum of the sizes of (encodings of) both Σ and ϕ. Model checkers based on interleaving are usually exponential in the size of the system and polynomial in the size of the formula [11, 13]. Let us examine which are the bounds that complexity theory imposes on the complexity of a model checker based on partial orders.

It is easy to show that our model checker cannot be polynomial in the size of the system unless P = PSPACE. The reachability problem for 1-safe systems (i.e., the problem of deciding, given a 1-safe system $\Sigma = (N, M_0)$ and a marking M of N, if M is reachable from M_0) is known to be PSPACE-complete [10]. This problem remains PSPACE-complete even if the marking to be tested only marks one place. To prove it, add a new transition having M as preset and the set $\{s\}$ as postset, where s is a new place. The marking M is reachable in the old system if and only if the marking $\{s\}$ is reachable in the new one (we use here that no transition has an empty postset). In turn, $\{s\}$ is reachable if and only if Σ satisfies $\Diamond s$. If our model checker were polynomial in the size of the system, then the reachability problem would be polynomial as well,

because the length of the formula is constant. Therefore, polynomiality in the size of the system implies P = PSPACE.

Let us now consider the complexity in the length of the formula. Our model checker should be polynomial in the size of the system for the first class of systems we consider in the introduction (Fig. 1). We show that it cannot be polynomial in the length of the formula unless P = NP by reducing SAT (satisfiability of formulas of propositional logic) to the model checking problem for this class of systems. Take a formula ϕ of propositional logic with variables $x_1, \ldots, x_n$. Take the net system of Fig. 1, and rename the places $s_1, \ldots, s_n$ as $x_1, \ldots, x_n$. Let Σ_n be the resulting system. Σ_n can be constructed in linear time in the length of ϕ, and so can the formula $\Diamond\phi$ of the logic L over Σ_n. It is immediate to see that $\Sigma_n \models \Diamond\phi$ if and only if ϕ is satisfiable. Therefore, a polynomial algorithm in both the size of the system and the length of the formula can be transformed into a polynomial algorithm for SAT.

These results show that it does not make much sense to compare interleaving and partial order model checkers on arbitrary systems or arbitrary formulas; on such a general problem we cannot expect to observe any interesting differences in the performance. It only makes sense to compare them on restricted classes of systems and restricted classes of formulas. We shall do so in the last chapter.

3.2. *A finite branching process adequate for model checking*

The maximal branching process β_m may be infinite, and therefore unsuitable as basis of a model checker. We define in this section a finite prefix of it. Then, we show how to solve the model checking problem on this finite prefix. Let us first make precise the notion of a prefix.

Definition 3.5 (*Prefixes of the maximal branching process* β_m). A branching process $\beta = (B, E, F, p)$ is a prefix of β_m if β is an approximation of β_m satisfying $B \subseteq B_m$ and $E \subseteq E_m$.

Notice that a prefix is determined by its set of events or its set of conditions, because any of them characterises the occurrence net of the process (the labelling of the prefix is just the restriction of p_m to the nodes of this occurrence net). We do not distinguish between the *Mark* mappings of the maximal branching process and the finite prefix.

The finite prefix must contain enough information to decide, for every property of the logic, if it holds or not. In particular, *every reachable marking of the system must be represented in the finite prefix* (more precisely, for every reachable marking M there must exist a configuration C of the finite prefix such that $Mark(C) = M$). This is necessary, because our logic can express the reachability of a marking; if some reachable marking M were not represented in the finite prefix, then we would not be able to decide whether the formula expressing the reachability of M holds.

The existence of a finite prefix satisfying this property is an immediate consequence of two facts: every reachable marking is represented by some cut of the maximal branching

process, and the set of reachable markings of a 1-safe system is finite. It is more difficult to give a reasonably efficient algorithm for its construction. McMillan provided in [29] an elegant solution for this problem. In order to describe it we need to introduce two notions: the set of causes of an event and the cut-off events of a branching process.

Definition 3.6 (*Set of causes of an event*). Let (B, E, F) be an occurrence net, and let $e \in E$ be an event. The set $[e] = \{e' \in E \mid e' \preceq e\}$ is the set of causes of e.

It is immediate to see that the set $[e]$ is a finite configuration for every event e.[6] Moreover, for every configuration C, either C does not contain e or it includes $[e]$.

The definition of a cut-off event is more compact if we introduce a new "event" symbol $\perp$ and define $[\perp]$ as the empty configuration. The reader may think of $\perp$ as a "virtual" event added to β_m, having no input conditions and the set of minimal elements of β_m as output conditions. Formally, the nature of $\perp$ is irrelevant, because we shall only use $[\perp]$.

Definition 3.7 (*Cut-off event*). An event $e \in E_m$ is a cut-off event if there exists a set of causes $[e']$ such that $|[e']| < |[e]|$ and $Mark([e']) = Mark([e])$.

If $Mark([e]) = M_0$ (the initial marking of Σ) for some event e, then e is a cut-off event, because $Mark([\perp]) = M_0$, and $0 = |[\perp]| < |[e]|$. This is the reason for the introduction of $\perp$.

In the branching process of Fig. 4, e_6 is a cut-off event, because $|[e_2]| < |[e_6]|$ and

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

The "continuation" of β_m from the cut $Cut([e])$ describes the future of the system from the marking $Mark([e])$. If e is a cut-off event, then we have $Mark([e]) = Mark([e'])$ for some event e' such that $|[e']| < |[e]|$. Since the possible behaviours of the system are completely determined by the initial marking, it is intuitively clear that the continuations of β_m from the cuts $Cut([e])$ and $Cut([e'])$ must be isomorphic. It is not difficult to formalise this statement. First, we have to make precise the notion of continuation.

Definition 3.8 (*The branching process* $\Uparrow c$). Let $\uparrow c = \{x \in B \cup E \mid c \preceq x\}$ where c be a cut of β_m. The branching process $\Uparrow c$ is defined as follows:

$$\Uparrow c = (B_m \cap \uparrow c, E_m \cap \uparrow c, F_m, p_m)$$

(where we identify F_m and p_m with their restriction to the nodes of $\Uparrow c$).

So $\Uparrow c$ contains the events and conditions "after" c. It is routine to check that $\Uparrow c$ is a branching process of the system $(N, p_m(c))$. The following proposition is also easy to prove:

[6] In [29], sets of causes are called *local configurations*.

Proposition 3.9. *If $Mark(C) = Mark(C')$ for two configurations C and C', then $\Uparrow Cut(C)$ is isomorphic to $\Uparrow Cut(C')$.*

We are now ready to define McMillan's finite prefix.

Definition 3.10 (*The finite prefix β_f*). Let E_f be the set of events of β_m given by:

$e \in E_f$ iff no event $e' \prec e$ is a cut-off event

β_f is the (unique) prefix of β_m having E_f as set of events.

An algorithm for the construction of the finite prefix β_f is described in [29]. The algorithm starts with the branching process containing no events, and adds events one at a time, in order increasing size of their sets of causes. For every new event e, the marking $Mark([e])$ is compared with the marking $Mark([e'])$ of each event e' which was added before e. If they are equal, then e is identified as a cut-off event, and its successor events are not explored (they are not part of the finite prefix). The algorithm terminates when no new event can be added.

The termination of the algorithm follows from the finiteness of the set of reachable markings of 1-safe systems. Let n be the number of reachable markings of Σ. Since cuts correspond to reachable markings, every event e such that $|[e]| > n$ must be a cut-off event, because $Mark([e]) = Mark([e'])$ must hold for some set of causes $[e'] \subset [e]$.

We refer the reader to [29] for a more detailed description of the algorithm. Here we only illustrate it by constructing the finite prefix for the system of Fig. 3. The events e_1, e_2, e_3, e_4 and e_5 of Fig. 5 are successively added; none of them is a cut-off event. Then, the event e_6 is added; we observe that $|[e_2]| < |[e_6]|$ and $Mark([e_2]) = \{s_3, s_4\} = Mark([e_6])$ (the cuts corresponding to $[e_2]$ and $[e_6]$ can be seen in the figure). So e_6 is identified as a cut-off event, and its successors are not considered for inclusion in the prefix. After that, the event e_7 is added to the prefix (e_7 is not a cut-off event). At this moment, no other event can be added (the only candidate would be the event e_9, but it is a successor of the cut-off event e_6) and therefore the procedure terminates. The output is the prefix having $\{e_1, \ldots, e_7\}$ as set of events.

We informally prove that the finite prefix β_f contains every reachable marking (in the next section we give a formal proof of a stronger statement, but this proof requires some preparation that might hide the simplicity of the main idea). Let M be an arbitrary reachable marking. Since every reachable marking is represented in β_m, there exists some configuration C of β_m such that $Mark(C) = M$. If C is a configuration of β_f, then we are done. Otherwise C is not included in E_f. By the definition of E_f, C contains some cut-off event e, and therefore $C \supseteq [e]$. By the definition of a cut-off event, there exists a set $[e']$ such that $|[e']| < |[e]|$ and $Mark([e']) = Mark([e])$. Since the branching processes $\Uparrow Cut([e])$ and $\Uparrow Cut([e'])$ are isomorphic, there exists a configuration $C' \supseteq [e']$ corresponding to the configuration C (in our example, taking $e = e_6$ and $e' = e_2$, the configuration corresponding to $\{e_2, e_3, e_4, e_5, e_6\}$ is $\{e_1, e_2\}$). Then, we also have $Mark(C') = M$; moreover, C' contains *less* events than C, because

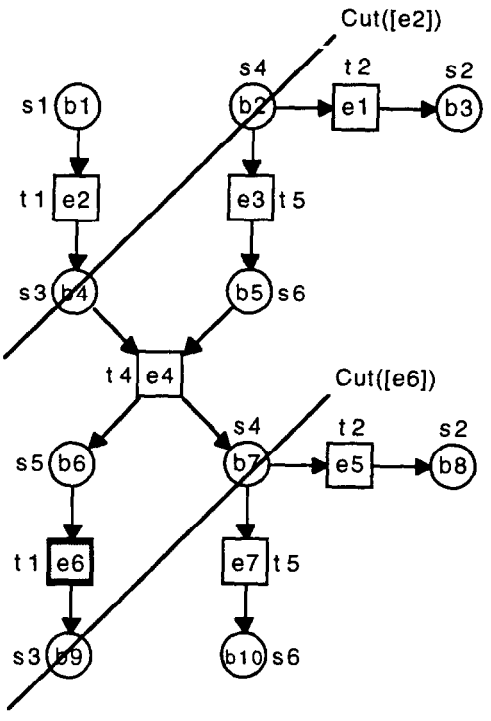

Fig. 5. The finite prefix for the 1-safe system of Fig. 3.

$[e']$ contains less events than $[e]$ (by the definition of a cut-off event). If C' is a configuration of β_f, we are done. Otherwise, C' contains a cut-off event, and the procedure can be iterated. Since every iteration produces a smaller configuration, the procedure terminates with a configuration of β_f, which finishes the proof.

We finish the section with some notations related to the finite prefix β_f that are frequently used in the sequel.

Notation 3.11 (*Notations related to the finite prefix* β_f).

- B_f and E_f denote the set of conditions and events of β_f, respectively. N_f denotes its underlying occurrence net.
- *Off* denotes the set of cut-off events of β_f (*Off* is also the set of minimal cut-off events with respect to the causal relation).
- Given a cut-off event e, there may exist several (but at least one) sets $[e']$ such that $Mark([e]) = Mark([e'])$ and $|[e']| < |[e]|$. We assume in the sequel that for each cut-off event e one of these sets has been fixed, and denote it by $[e^0]$.
- $\mathcal{F}$ denotes the set of configurations of β_f.
- $\mathcal{D}$ denotes the set of maximal configurations of β_f with respect to set inclusion.

In order to verify that the event e is a cut-off event, McMillan's algorithm has to find some set $[e']$ satisfying $Mark([e]) = Mark([e'])$ and $|[e']| < |[e]|$. In practice, $[e^0]$

is taken as this $[e']$; so the computation of $[e^0]$ for each cut-off event takes place "on the fly" during the construction of the finite prefix.

3.3. Shifts

Given a cut-off event e, the branching processes $\Uparrow Cut([e^0])$ and $\Uparrow Cut([e])$ are isomorphic. In fact, the branching process $\Uparrow Cut([e])$ can be thought to be $\Uparrow Cut([e^0])$ "shifted forward". For a configuration C containing $[e^0]$ there exists a corresponding configuration containing $[e]$, obtained by "shifting C forward". In the next definition we formalise this idea of a shift.

Definition 3.12 (*Shift of a configuration*). Let e be a cut-off event of β_m and let I_e be the isomorphism from $\Uparrow Cut([e^0])$ to $\Uparrow Cut([e])$.

Let C be a configuration of β_m. The e-shift of C, denoted by $\mathcal{S}_e(C)$, is the following configuration:

$$\mathcal{S}_e(C) = \begin{cases} C & \text{if } [e^0] \not\subseteq C \\ [e] \cup I_e(C \setminus [e^0]) & \text{if } [e^0] \subseteq C \end{cases}$$

Notice that the isomorphism I_e depends on our choice of e^0, so it would be more correct to write $I_{(e,e^0)}$. We choose not to do it in order to keep the notation simple, and because we assume that e^0 has been fixed for each e. In the example of Fig. 5, we fix $e_6^0 = e_2$ (in this particular case, this is the only possibility).

It is immediate to prove that $\mathcal{S}_e(C)$ is a configuration, and therefore well defined. In Fig. 4 we have

$$\mathcal{S}_{e_6}(\{e_1, e_2\}) = [e_6] \cup I_{e_6}(\{e_1\}) = \{e_2, e_3, e_4, e_6\} \cup \{e_5\}$$

as expected.

Shifts have the following properties:

Proposition 3.13. *Let e be a cut-off event and let C, C' be two configurations of β_m.*

(1) *$\mathcal{S}_e$ is injective and monotonic.*

(2) *$Mark(C) = Mark(\mathcal{S}_e(C))$*

(3) *$|\mathcal{S}_e(C)| \geqslant |C|$, and the equality holds only if $\mathcal{S}_e(C) = C$.*

(4) *If $e \in C$, then $\mathcal{S}_e^{-1}(C)$ exists.*

Proof. (1) and (2) follow immediately from the fact that I_e is an isomorphism.

(3) It suffices to show that if $[e^0] \not\subseteq C$, then $|\mathcal{S}_e(C)| > |C|$. We have

$$\begin{aligned} & |\mathcal{S}_e(C)| \\ = & \{\text{Definition of } \mathcal{S}_e\} \\ & |[e] \cup I_e(C \setminus [e^0])| \end{aligned}$$

$$= \{[e] \cap I_e(C \setminus [e^0]) = \emptyset\}$$

$$|[e]| + |I_e(C \setminus [e^0])|$$

$$> \{|[e]| > |[e^0]| \text{ by the definition of cut-off event}\}$$

$$|[e^0]| + |C \setminus [e^0]|$$

$$= \{[e^0] \subseteq C\}$$

$$|C|$$

(4) If $e \in C$, then $[e] \subseteq C$, and $\mathcal{S}_e^{-1}(C) = [e^0] \cup I_e^{-1}(C \setminus [e])$. □

Consider the set of configurations of the maximal branching process corresponding to the same reachable marking M. Such a set can be infinite, as it happens for instance in the example of Fig. 4 with the configurations corresponding to the marking $\{s_3, s_6\}$. By Proposition 3.13(2), the image of a configuration of this set under an e-shift is another configuration of the same set. The same holds for e-shift inverses (in case the configuration contains the event e). This notion of invariance under shifts is captured in the next definition.

Definition 3.14 (*Invariance under shifts*). A set of configurations $\mathcal{C}$ is invariant under shifts (or just invariant) if for every cut-off event e and every configuration C, $C \in \mathcal{C}$ iff $\mathcal{S}_e(C) \in \mathcal{C}$.

We show that the set of configurations that satisfy an arbitrary formula of the logic is invariant. Since, given a marking M, there exists a formula expressing $Mark(C) = M$, this result generalises the invariance of the set of configurations corresponding to the same marking.

Lemma 3.15 (Invariance of $Sat(\phi)$ under shifts). *For every formula ϕ, $Sat(\phi)$ is invariant under shifts.*

Proof. Let e and C be a cut-off event and a configuration of β_m. We show that $C \models \phi$ iff $\mathcal{S}_e(C) \models \phi$. By Proposition 3.9(2), we have $Mark(C) = Mark(\mathcal{S}_e(C))$. So $\Uparrow Cut(C)$ is isomorphic to $\Uparrow Cut(C')$. It follows easily from the semantics of L that whether a configuration C satisfies a formula ϕ or not depends only on $\Uparrow Cut(C)$. Therefore, C and $\mathcal{S}_e(C)$ satisfy the same formulas of L. □

We are now ready to show that $Sat(\phi)$ can be generated by the subset of $Sat(\phi)$ formed by the configurations contained in the finite prefix.

Definition 3.16 (*The set $Sat_f(\phi)$*). Let ϕ be a formula. Then $Sat_f(\phi) = Sat(\phi) \cap \mathcal{F}$. (Recall that $\mathcal{F}$ denotes the set of configurations of the finite prefix β_f.)

Definition 3.17 (*The mapping* $\mathcal{S}$). Let $\mathcal{C}$ be a set of configurations of β_m. The set of configurations $\mathcal{S}_e(\mathcal{C})$ is given by:

$$\mathcal{S}_e(\mathcal{C}) = \{\mathcal{S}_e(C) \mid C \in \mathcal{C}\}$$

The set of configurations $\mathcal{S}(\mathcal{C})$ is given by:

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

(recall that *Off* denotes the set of cut-off events of β_f).

We define:

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

Notice that $\mathcal{S}$ is defined on sets of configurations. It follows easily from this definition that $\mathcal{S}$ is a monotonic function on the complete partial order of the sets of configurations of β_m, where the order is set inclusion (notice that this partial order is different from the partial order of configurations). It is easy to see that $\mu\mathcal{S}.\mathcal{C}$ is the least fixpoint of $\mathcal{S}$ containing $\mathcal{C}$, which explains the notation we have chosen.

Theorem 3.18. *For every formula* ϕ, $Sat(\phi) = \mu\mathcal{S}.Sat_f(\phi)$.

Proof. We prove separately both inclusions.

($\subseteq$) Let $C \in Sat(\phi)$, and consider two cases:

Case 1. If C does not contain any cut-off event, then C is a configuration of β_f. So $C \in Sat_f(\phi)$, and we are done.

Case 2. If C does contain a cut-off event e, then $\mathcal{S}_e^{-1}(C)$ exists and $\mathcal{S}_e^{-1}(C) \in Sat(\phi)$ by the invariance of $Sat(\phi)$. Moreover, $|\mathcal{S}_e^{-1}(C)| < |C|$. If $\mathcal{S}_e^{-1}(C)$ does not contain any cut-off event, then $\mathcal{S}_e^{-1}(C) \in Sat_f(\phi)$, which implies $C \in \mathcal{S}(Sat_f(\phi))$. Else, $\mathcal{S}_e^{-1}(C)$ contains a cut-off event e' and $\mathcal{S}_{e'}^{-1}(\mathcal{S}_e^{-1}(C))$ exists. If $\mathcal{S}_{e'}^{-1}(\mathcal{S}_e^{-1}(C))$ contains no cut-off event, then we get $C \in \mathcal{S}^2(Sat_f(\phi))$. This procedure can be iterated, and eventually terminates for some number n, because C is finite and each application of one of the inverses $\mathcal{S}_e^{-1}$ decreases the number of events. So $C \in \mathcal{S}^n(Sat_f(\phi))$ for some n.

($\supseteq$) Since $Sat_f(\phi) \subseteq Sat(\phi)$ and $Sat(\phi)$ is invariant, we have $\mathcal{S}_e^n(Sat_f(\phi)) \subseteq Sat(\phi)$ for every $n \geqslant 0$. □

An important consequence of Theorem 3.18 is that $Sat(\phi)$ is empty if and only if $Sat_f(\phi)$ is empty. That is, some configuration satisfies ϕ iff some configuration of the finite prefix satisfies it. Therefore, the model checking problem reduces to the problem of deciding if $Sat_f(\phi)$ is empty.

Our problem is to find an efficient algorithm for emptiness. Since the finite prefix contains finitely many configurations, the obvious algorithm is to exhaustively check them all. However, this corresponds to checking all the reachable markings of the

system, which would cancel any possible advantage of our approach with respect to the interleaving-based model checkers. We show that there is a much more efficient algorithm to compute the *maximal* elements of $Sat_f(\phi)$ with respect to set inclusion. This is a fundamental idea of our approach: since we wish to avoid the enumeration of the states of the system — the finite prefix does not even have an explicit representation of global states; they are "embedded" in it — we cannot perform any search of the state space. Instead, we develop algorithms to identify particularly important states, namely the maximal elements of $Sat_f(\phi)$.

It is convenient to introduce a notation for the set of maximal elements of $Sat_f(\phi)$.

Definition 3.19 (*Last sets of configurations*). Let $max\{C\}$ denote the set of maximal elements of a set of configurations C with respect to set inclusion. We define $Last(\phi) = max\{Sat_f(\phi)\}$ (*Last* is short for *Largest*).

Since $Last(\phi)$ is empty if and only $Sat_f(\phi)$ is empty, the model checking problem reduces to deciding, for an arbitrary formula ϕ, if $Last(\phi) = \emptyset$ or not.

3.4. Compositional equations

In this section we obtain compositional equations for $Last(\phi)$, where ϕ is a formula in a certain normal form. We use a generalization of the conjunctive normal form of propositional logic.

Definition 3.20 (*Normal form*). A formula ϕ is in normal form if it is generated by the following grammar:

$$\gamma ::= \textbf{true} \mid \textbf{false} \mid s \mid \neg s \mid \gamma \wedge \gamma$$
$$\phi ::= \gamma \mid \phi \wedge \Diamond \phi \mid \phi \wedge \neg \Diamond \phi$$

In the sequel, as was done in this definition, the symbol γ is used to denote conjunctions of literals. A typical formula which is not in normal form is $s_1 \vee s_2$.

We prove that every formula is equivalent (denoted by $\equiv$) to a disjunction of formulas in normal form.

Proposition 3.21. *Let ϕ be a formula. There exist formulas $\phi_1, \ldots, \phi_n$ in normal form such that*

$$\phi \equiv \bigvee_{i=1}^{n} \phi_i$$

Proof. Push in negations through disjunctions and conjunctions using the DeMorgan laws. Then, move out disjunctions through conjunctions (using the distributive law) and $\Diamond$ operators (using $\Diamond(\phi_1 \vee \phi_2) \equiv \Diamond\phi_1 \vee \Diamond\phi_2$). □

By this proposition, we have

$$Sat(\phi) = Sat\left(\bigvee_{i=1}^{n} \phi_i\right) = \bigcup_{i=1}^{n} Sat(\phi_i)$$

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 (because disjunctions must be moved out through conjunctions). However, this problem does not have much relevance. Formulas are usually short (at least compared with the finite prefix), and many formulas of interest, e.g. those expressing reachability, mutual exclusion, liveness and cyclicity, are in normal form.[7] Moreover, as shown before, if P $\neq$ NP then we cannot have a polynomial algorithm in the length of the formula.

It is easier to give compositional equations for a generalized *Last* set which has as parameters not only a formula ϕ, but also two sets of configurations. The generalization makes use of the following relation:

Definition 3.22 (*The relation* $\ll$). Let C be a configuration and let $\mathcal{C}$ be a set of configurations. We say $C \ll \mathcal{C}$ (read "C is less than $\mathcal{C}$") if there exists some configuration $C' \in \mathcal{C}$ such that $C \subseteq C'$.

Let $\mathcal{C}_1$, $\mathcal{C}_2$ be two sets of configurations. We say $\mathcal{C}_1 \ll \mathcal{C}_2$ if $C_1 \ll \mathcal{C}_2$ for every configuration $C_1 \in \mathcal{C}_1$.

The relation $\ll$ has the following fundamental property:

Lemma 3.23 (Fundamental property of the relation $\ll$). *Let C be a configuration and let ϕ be a formula. Then, $C \models \Diamond\phi$ iff $C \ll Sat(\phi)$.*

Proof. Follows immediately from the definitions. □

After these preliminaries, we can define the generalized *Sat* and *Last* sets.

Definition 3.24 (*Generalised Sat and Last sets*). Let ϕ be a formula and let $\mathcal{C}_1$, $\mathcal{C}_2$ be two sets of configurations. The set of configurations $Sat(\mathcal{C}_1, \phi, \mathcal{C}_2)$ is defined as follows:

$$C \in Sat(\mathcal{C}_1, \phi, \mathcal{C}_2) \text{ iff } C \models \phi \text{ and } C \not\ll \mathcal{C}_1 \text{ and } C \ll \mathcal{C}_2$$

Let $\mathcal{C}_1$, $\mathcal{C}_2 \subseteq \mathcal{F}$ be two sets of configurations. We define:

$$Sat_f(\mathcal{C}_1, \phi, \mathcal{C}_2) = Sat(\mathcal{C}_1, \phi, \mathcal{C}_2) \cap \mathcal{F}$$

$$Last(\mathcal{C}_1, \phi, \mathcal{C}_2) = max\{Sat_f(\mathcal{C}_1, \phi, \mathcal{C}_2)\}.$$

[7] Once the derived operator □ is substituted by its definition.

In words, $Sat(\mathcal{C}_1, \phi, \mathcal{C}_2)$ is the set of configurations that satisfy ϕ, are included in some configuration of the set $\mathcal{C}_2$, and are not included in any of the configurations of the set $\mathcal{C}_1$.

Notice that no configuration is less than the empty set of configurations; on the other hand, every configuration is less than the set of maximal configurations of β_m. Therefore, when $\mathcal{C}_1$ is the empty set and $\mathcal{C}_2$ the set of maximal configurations, $Sat(\mathcal{C}_1, \phi, \mathcal{C}_2)$ is just the set of configurations that satisfy ϕ, i.e., the set $Sat(\phi)$. Similarly, we have $Sat_f(\phi) = Sat_f(\emptyset, \phi, \mathcal{D})$, because every configuration of the finite prefix β_f is less than its set of maximal configurations, which is the set $\mathcal{D}$. Finally, $Last(\phi) = Last(\emptyset, \phi, \mathcal{D})$

Guided by the grammar of the formulas in normal form, our first task consists in expressing both $Sat(\mathcal{C}_1, \phi \wedge \Diamond\psi, \mathcal{C}_2)$ and $Sat(\mathcal{C}_1, \phi \wedge \neg\Diamond\psi, \mathcal{C}_2)$ as a function of $Sat(\mathcal{C}_1', \phi, \mathcal{C}_2')$ and $Sat(\mathcal{C}_1'', \psi, \mathcal{C}_2'')$ for well chosen sets of configurations $\mathcal{C}_1'$, $\mathcal{C}_2'$, $\mathcal{C}_1''$ and $\mathcal{C}_2''$. For this purpose, we introduce a new operator.

Definition 3.25 (*The operator* $\bigtriangledown$). Let $\mathcal{C}_1$, $\mathcal{C}_2$ be two sets of configurations. We define

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

Notice that $\mathcal{C}_1 \bigtriangledown \mathcal{C}_2$ is a set of configurations because the set of configurations is closed under intersection. Moreover, $\bigtriangledown$ is associative and commutative. The following property is easy to prove:

Lemma 3.26 (Fundamental property of $\bigtriangledown$). *Let C be a configuration and let $\mathcal{C}_1$, $\mathcal{C}_2$ be two sets of configurations. Then, $C \ll \mathcal{C}_1 \bigtriangledown \mathcal{C}_2$ iff $C \ll \mathcal{C}_1$ and $C \ll \mathcal{C}_2$.*

We also need an operator capturing that a configuration satisfies $C \ll \mathcal{C}_1$ and $C \ll \mathcal{C}_2$. In this case, we do not have to introduce any new operator, because union of sets of configurations does the job.

Lemma 3.27 (Fundamental property of $\cup$ (on sets of configurations)). *Let C be a configuration and let $\mathcal{C}_1$, $\mathcal{C}_2$ be two sets of configurations. Then, $C \not\ll \mathcal{C}_1 \cup \mathcal{C}_2$ iff $C \not\ll \mathcal{C}_1$ and $C \not\ll \mathcal{C}_2$.*

Notice the lack of symmetry between both cases (the two operators we need are neither union nor intersection, nor $\bigtriangledown$ and its dual operator). This is not surprising, because the symmetry between conjunction and disjunction was broken in our normal form, and our logic is not symmetric with respect to future and past.

Theorem 3.28 (Compositional equations for *Sat*). *Let $\mathcal{C}_1$, $\mathcal{C}_2$ be two sets of configurations of β_m, and let ϕ, ψ be two formulas in normal form. We have:*

$$Sat(\mathcal{C}_1, \phi \wedge \Diamond\psi, \mathcal{C}_2) = Sat(\mathcal{C}_1, \phi, \mathcal{C}_2 \bigtriangledown Sat(\psi))$$

$$Sat(\mathcal{C}_1, \phi \wedge \neg\Diamond\psi, \mathcal{C}_2) = Sat(\mathcal{C}_1 \cup Sat(\psi), \phi, \mathcal{C}_2)$$

Proof. We prove the second equation. The first is similar.

($\subseteq$) Let $C \in Sat(\mathcal{C}_1, \phi \wedge \neg\Diamond\psi, \mathcal{C}_2)$. By Definition 3.24, we have $C \models \phi \wedge \neg\Diamond\psi$, $C \not\ll \mathcal{C}_1$ and $C \ll \mathcal{C}_2$. By the fundamental property of $\ll$, $C \models \phi$ and $C \not\ll Sat(\psi)$. By the fundamental property of $\cup$, $C \not\ll \mathcal{C}_2 \cup Sat(\psi)$. So $C \in Sat(\mathcal{C}_1 \cup Sat(\psi), \phi, \mathcal{C}_2)$.

($\supseteq$): Let $C \in Sat(\mathcal{C}_1 \cup Sat(\psi), \phi, \mathcal{C}_2)$. By Definition 3.24, we have $C \models \phi$, $C \not\ll \mathcal{C}_1 \cup Sat(\psi)$ and $C \ll \mathcal{C}_2$. By the fundamental property of $\cup$, $C \not\ll \mathcal{C}_1$ and $C \not\ll Sat(\psi)$. By the fundamental property of $\ll$, $C \not\models \Diamond\psi$. So $C \in Sat(\mathcal{C}_1, \phi \wedge \neg\Diamond\psi, \mathcal{C}_2)$. □

After an exhaustive application of these two equations, we obtain an expression for $Sat(\mathcal{C}_1, \phi, \mathcal{C}_2)$ in which no $\Diamond$-symbol appears. Specialising $\mathcal{C}_1$ to the empty set and $\mathcal{C}_2$ to the set of maximal configurations of β_m, we get an expression for $Sat(\phi)$ with the same property.

We now derive similar compositional equations for $Sat_f(\phi)$. The equations only use sets of configurations of the finite prefix. Notice in particular that, by the definition of $\triangledown$, for any set of configurations $\mathcal{C}$ the set $\mathcal{C} \triangledown \{E_f\}$ only contains configurations of the finite prefix.

Theorem 3.29 (Compositional equations for Sat_f). *Let $\mathcal{C}_1, \mathcal{C}_2 \subseteq \mathcal{F}$ be two sets of configurations of the finite prefix. Let $\mathcal{C} = \mu\mathcal{S}.Sat_f(\psi) \triangledown \{E_f\}$. We have:*

$$Sat_f(\mathcal{C}_1, \phi \wedge \Diamond\psi, \mathcal{C}_2) = Sat_f(\mathcal{C}_1, \phi, \mathcal{C}_2 \triangledown \mathcal{C})$$
$$Sat_f(\mathcal{C}_1, \phi \wedge \neg\Diamond\psi, \mathcal{C}_2) = Sat_f(\mathcal{C}_1 \cup \mathcal{C}, \phi, \mathcal{C}_2)$$

Proof. We prove the first equality. The proof of the second is similar.

Since $\mathcal{C}_2$ is a set of configurations of the finite prefix, all the elements of $Sat(\mathcal{C}_1, \phi \wedge \Diamond\psi, \mathcal{C}_2)$ are configurations of the finite prefix as well (they are contained in some configuration of $\mathcal{C}_2$). Therefore, we have

$$Sat_f(\mathcal{C}_1, \phi \wedge \Diamond\psi, \mathcal{C}_2) = Sat(\mathcal{C}_1, \phi \wedge \Diamond\psi, \mathcal{C}_2)$$

We prove

$$Sat(\mathcal{C}_1, \phi, \mathcal{C}_2 \triangledown Sat(\psi)) = Sat_f(\mathcal{C}_1, \phi, \mathcal{C}_2 \triangledown \mathcal{C})$$

which, by the equations for Sat, implies the result.

Since $\mathcal{C}_2 \triangledown Sat(\psi)$ is a set of configurations of the finite prefix, it suffices to prove $\mathcal{C}_2 \triangledown Sat(\psi) = \mathcal{C}_2 \triangledown \mathcal{C}$. By the definition of $\mathcal{C}$, and since $\mu\mathcal{S}.Sat_f(\psi) = Sat(\psi)$ (Theorem 3.18), this reduces in turn to showing

$$\mathcal{C}_2 \triangledown Sat(\psi) = \mathcal{C}_2 \triangledown (Sat(\psi) \triangledown \{E_f\})$$

To prove it, use $\mathcal{C}_2 \triangledown \{E_f\} = \mathcal{C}_2$, which holds because $C \subseteq E_f$ for every $C \in \mathcal{C}_2$. □

The configurations of the set $\mu\mathcal{S}.Sat_f(\psi) \triangledown \{E_f\}$ are contained in E_f by definition, and therefore have a bounded size. This would be no advantage if in order to compute $\mu\mathcal{S}.Sat_f(\psi) \triangledown \{E_f\}$ we first had to compute $\mu\mathcal{S}.Sat_f(\psi)$. However, as we shall show,

this is not the case: $\mu\mathcal{S}.Sat_f(\psi) \triangledown \{E_f\}$ can be directly computed on the finite prefix. The next section shows how to do it.

Finally, we obtain equations for $Last(\mathcal{C}_1, \phi, \mathcal{C}_2)$, the set of maximal configurations of $Sat_f(\mathcal{C}_1, \phi, \mathcal{C}_2)$. The correctness proofs of the equations make use of the following elementary properties of the *max* operator (the proof is omitted):

Lemma 3.30 (Some properties of *max*).
(1) $max\{max\{\mathcal{C}\}\} = max\{\mathcal{C}\}$.
(2) $max\{max\{\bigcup_{i\in I} \mathcal{C}_i\}\} = max\{\bigcup_{i\in I} max\{\mathcal{C}_i\}\}$.
(3) $max\{max\{\triangledown_{i\in I}\mathcal{C}_i\}\} = max\{\triangledown_{i\in I} max\{\mathcal{C}_i\}\}$.

Observe that *max* and union, or *max* and $\triangledown$, do not commute in general.

Theorem 3.31 (Compositional equations for *Last*). *Let $\mathcal{C}_1, \mathcal{C}_2 \subseteq \mathcal{F}$ be two sets of configurations of the finite prefix.*
Let $\mathcal{C} = \mu\mathcal{S}.Last(\psi) \triangledown \{E_f\}$. We have:

$$Last(\mathcal{C}_1, \phi \wedge \Diamond\psi, \mathcal{C}_2) = Last(\mathcal{C}_1, \phi, max\{\mathcal{C}_2 \triangledown \mathcal{C}\})$$
$$Last(\mathcal{C}_1, \phi \wedge \neg\Diamond\psi, \mathcal{C}_2) = Last(max\{\mathcal{C}_1 \cup \mathcal{C}\}, \phi, \mathcal{C}_2)$$

Proof. We prove the first equality. The proof of the second is similar.
We claim $max\{\mu\mathcal{S}.Sat_f(\psi)\} = max\{\mu\mathcal{S}.Last(\psi)\}$.

$$\begin{aligned}
& max\{\mu\mathcal{S}.Sat_f(\psi)\} \\
&= \{\text{definition of } \mu\mathcal{S}.\mathcal{C}\text{, properties of } max\} \\
& max\{\bigcup_{n\geq 0} max\{\mathcal{S}^n(Sat_f(\psi))\}\} \\
&= \{\text{monotonicity of } \mathcal{S}_e \text{ for every cut-off event } e\} \\
& max\{\bigcup_{n\geq 0} \mathcal{S}^n(max\{Sat_f(\psi)\})\} \\
&= \{\text{ definitions of } Last(\psi) \text{ and } \mu\mathcal{S}.\mathcal{C} \text{ }\} \\
& max\{\mu\mathcal{S}.Last(\psi)\}
\end{aligned}$$

Now, we have

$$\begin{aligned}
& Last(\mathcal{C}_1, \phi \wedge \Diamond\psi, \mathcal{C}_2) \\
&= \{\text{definition of } Last\} \\
& max\{Sat_f(\mathcal{C}_1, \phi \wedge \Diamond\psi, \mathcal{C}_2)\} \\
&= \{\text{equation for } Sat_f\text{, definition of } \mathcal{C}\text{, associativity of } \triangledown\} \\
& max\{Sat_f(\mathcal{C}_1, \phi, \mathcal{C}_2 \triangledown \mu\mathcal{S}.Sat_f(\psi) \triangledown \{E_f\})\} \\
&= \{\mathcal{C} \ll \mathcal{C} \text{ iff } \mathcal{C} \ll max\{\mathcal{C}\}\} \\
& max\{Sat_f(\mathcal{C}_1, \phi, max\{\mathcal{C}_2 \triangledown \mu\mathcal{S}.Sat_f(\psi) \triangledown \{E_f\}\})\} \\
&= \{\text{properties of } max\text{, result above}\}
\end{aligned}$$

$max\{Sat_f(C_1, \phi, max\{C_2 \triangledown \mu\mathcal{S}.Last(\psi) \triangledown \{E_f\}\})\}$
$= \{\text{definition of } Last\}$
$Last(C_1, \phi, max\{C_2 \triangledown \mu\mathcal{S}.Last(\psi) \triangledown \{E_f\}\})$ □

This is the result we have been aiming at. Let us show that it reduces the problem of deciding the emptyness of $Last(\phi)$ for a formula in normal form to the following two problems:

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

If ϕ is a conjunction of literals, then we are done by (2), because $Last(\phi) = Last(\emptyset, \phi, \mathcal{D})$. Now, assume we want to compute $Last(\phi \wedge \neg\Diamond\psi)$. By induction hypothesis, we can compute $Last(\psi)$. By (1), we can compute the set $\mathcal{C} = \mu\mathcal{S}.Last(\psi) \triangledown \{E_f\}$. By the induction hypothesis again, we can compute $Last(\mathcal{C}, \phi, \mathcal{D})$. Since $max\{\emptyset \cup \mathcal{C}\} = \mathcal{C}$, we have $Last(\mathcal{C}, \phi, \mathcal{D}) = Last(\phi \wedge \neg\Diamond\psi)$ by the second equation of Theorem 3.31. Similar reasoning shows that we can also compute $Last(\phi \wedge \Diamond\psi)$.

We formalise this argument in Section 6 by giving a recursive procedure for the computation of $Last(\phi)$. Before that, we provide algorithms for (1) and (2).

3.5. Algorithms

An algorithm to compute $\mu\mathcal{S}.\mathcal{C} \triangledown \{E_f\}$

We solve this problem in two steps. First, we show how to compute the configuration $\mathcal{S}_e(C) \cap E_f$ for a configuration $C \subseteq E_f$ and a cut-off event e. Using this, it will be easy to give an algorithm for the computation of $\mu\mathcal{S}.\mathcal{C} \triangledown \{E_f\}$.

By the definition of e-shift, we have:

$$\mathcal{S}_e(C) \cap E_f = \begin{cases} C & \text{if } [e^0] \not\subseteq C \\ ([e] \cup I_e(C \setminus [e^0])) \cap E_f & \text{otherwise} \end{cases}$$

where I_e is the isomorphism from $\Uparrow Cut([e^0])$ onto $\Uparrow Cut([e])$. The configurations $[e]$ and $[e^0]$ are obtained "on the fly" during the construction of the finite prefix. Therefore, our problem consists in computing $I_e(C \setminus [e^0]) \cap E_f$, which is easy once the pairs $(x, I_e(x))$ such that $I_e(x)$ is a node of the finite prefix are known.

Algorithm 3.32.
Input: The set $P_0 = \{(b, b') \mid b \in Cut([e^0]) \wedge b' \in Cut([e]) \wedge p_m(b) = p_m(b')\}$.
Output: The set P of pairs $(x, I_e(x))$ such that $I_e(x) \in B_f \cup E_f$.
begin
$P := P_0$; $Q := P_0$;
while $Q \neq \emptyset$ **do**
choose $(x, x') \in Q$;
for every $y' \in x'^{\bullet}$ **do**

```
      if ∀z' ∈ •y' ∃z (z, z') ∈ Q then
        let y be the unique node of x• such that p_m(y) = p_m(y');
        P := P ∪ {(y, y')}; Q := Q ∪ {(y, y')}
      endif
    endfor
  Q := Q \ {(x, x')}
  endwhile
end
```

The algorithm terminates because, (x, x') is eventually a pair such that x' is a maximal node of the finite prefix, and therefore $x'^{\bullet}$ is empty. In this case, the algorithm just removes (x, x') from Q without adding any new element.

Let us use the algorithm to compute the pairs $(x, I_{e_6}(x))$ in the example of Fig. 5. In this case, $[e_6^0] = [e_2]$, $Cut([e_2]) = \{b_2, b_4\}$, and $Cut([e_6]) = \{b_7, b_9\}$.

We have $P_0 = \{(b_2, b_7), (b_4, b_9)\} = Q$. We choose (b_2, b_7) out of Q and consider the output events of b_7, which are e_7 and e_5. The algorithm looks for appropriate output events of b_2 matching these two, in this case e_1 and e_3, adds (e_1, e_5), (e_3, e_7) as new pairs to P and Q and removes (b_2, b_7) from Q. The algorithm continues until all elements of β_f after $Cut([e_6])$ have been matched.

We now define the following finite version of the mapping $\mathcal{S}$.

Definition 3.33 (*Finite version $\mathcal{S}_f$ of the mapping $\mathcal{S}$*). Let $\mathcal{C}$ be a set of configurations of the finite prefix β_f. We define:

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

Also, we define

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

Notice that $\mathcal{S}_e(\mathcal{C}) \triangledown \{E_f\} = \{\mathcal{S}_e(C) \cap E_f \mid C \in \mathcal{C}\}$, and that $\mathcal{S}_f$ is a monotonic mapping. Since $\mathcal{S}_f^n(\mathcal{C})$ is a set of configurations of the finite prefix for every n, and the finite prefix contains finitely many configurations, there exists an integer k such that $\mathcal{S}_f^k(\mathcal{C}) = \mathcal{S}_f^{k+1}(\mathcal{C})$. We then have $\mu\mathcal{S}_f.\mathcal{C} = \mathcal{S}_f^k(\mathcal{C})$.

Let us calculate $\mu\mathcal{S}_f.\{\{e_2\}\}$ in the example of Fig. 5. In this case, e_6 is the only cut-off event.

$$\mathcal{S}_f^0(\{\{e_2\}\}) = \{\{e_2\}\}$$
$$\mathcal{S}_f^1(\{\{e_2\}\}) = \{\{e_2\}, \mathcal{S}_{e_6}(\{\{e_2\}\}) \cap E_f\} = \{\{e_2\}, \{e_2, e_3, e_4\}\}$$
$$\mathcal{S}_f^2(\{\{e_2\}\}) = \{\{e_2\}, \{e_2, e_3, e_4\}\}$$

So $k = 1$, and $\mu\mathcal{S}_f.\{\{e_2\}\} = \{\{e_2\}, \{e_2, e_3, e_4\}\}$.

Finally, the following theorem shows that $\mu\mathcal{S}_f.\mathcal{C}$ is all we need, because it is equal to the set $\mu\mathcal{S}.\mathcal{C} \triangledown \{E_f\}$.

Theorem 3.34. *For every set* $\mathcal{C} \subseteq \mathcal{F}$ *and every* $n \geqslant 0$, $\mathcal{S}^n(\mathcal{C}) \triangledown \{E_f\} = \mathcal{S}_f^n(\mathcal{C})$. *In particular, we have* $\mu\mathcal{S}.\mathcal{C} \triangledown \{E_f\} = \mu\mathcal{S}_f.\mathcal{C}$.

Proof. We proceed by induction on n.

Base. $n = 0$. We have to prove $\mathcal{C} \triangledown \{E_f\} = \mathcal{C}$. This follows easily from $\mathcal{C} \subseteq \mathcal{F}$, which implies $C \subseteq E_f$ for every $C \in \mathcal{C}$.

Step. $n \to n + 1$. We have $\mathcal{S}_e(C) \cap E_f = \mathcal{S}_e(C \cap E_f) \cap E_f$ for every cut-off event e and every configuration C (because the events of C not contained in E_f are shifted out of E_f). So for every set of configurations $\mathcal{C}$

$$\mathcal{S}(\mathcal{C}) \triangledown \{E_f\} = \mathcal{S}(\mathcal{C} \triangledown \{E_f\}) \triangledown \{E_f\} \qquad (*)$$

Therefore

$$\begin{aligned}
&\mathcal{S}^{n+1}(\mathcal{C}) \triangledown \{E_f\} \\
&= \{\text{definition}\} \\
&\mathcal{S}(\mathcal{S}^n(\mathcal{C})) \triangledown \{E_f\} \\
&= \{\text{equation } (*)\} \\
&\mathcal{S}(\mathcal{S}^n(\mathcal{C}) \triangledown \{E_f\}) \triangledown \{E_f\} \\
&= \{\text{induction hypothesis}\} \\
&\mathcal{S}(\mathcal{S}_f^n(\mathcal{C})) \triangledown \{E_f\} \\
&= \{\triangledown \text{ distributes over union}\} \\
&\mathcal{S}_f^{n+1}(\mathcal{C}) \qquad \square
\end{aligned}$$

An algorithm to compute $Last(\mathcal{C}_1, \gamma, \mathcal{C}_2)$

We provide an algorithm to compute $Last(\mathcal{C}_1, \gamma, \mathcal{C}_2)$, where γ is a conjunction of literals and $\mathcal{C}_1$, $\mathcal{C}_2$ are two sets of configurations of the finite prefix. We first obtain some preliminary results. Let us examine $Last(\mathcal{C}_1, \gamma, \mathcal{C}_2)$ in more detail.

$$\begin{aligned}
&Last(\mathcal{C}_1, \gamma, \mathcal{C}_2) \\
&= \{\text{definition of } Last \text{ and } Sat_f\} \\
&max\{C \mid C \models \gamma \wedge C \not\ll \mathcal{C}_1 \wedge C \ll \mathcal{C}_2\} \\
&= \{\text{definition of } C \ll \mathcal{C}_2, \text{ properties of } max\} \\
&max\{\bigcup_{C_2 \in \mathcal{C}_2} max\{C \mid C \models \gamma \wedge C \not\ll \mathcal{C}_1 \wedge C \subseteq C_2\}\} \\
&= \{\text{definition of } Last \text{ and } Sat\} \\
&max\{\bigcup_{C_2 \in \mathcal{C}_2} Last(\mathcal{C}_1, \gamma, \{C_2\})\}
\end{aligned}$$

Therefore, it suffices to provide an algorithm for the particular instances of the problem having the form $Last(\mathcal{C}_1, \gamma, \{C_2\})$. Furthermore, we have the following equivalence:

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

Since $C \not\ll C_1$ can be checked using the definition (it suffices to see whether C is contained in some configuration of the set C_1), we consider in the sequel the problem of computing $Last(\emptyset, \gamma, \{C_2\})$.

We prove that the set $Last(\emptyset, \gamma, \{C_2\})$ contains at most one configuration (i.e., it is either the empty set or a singleton). This result is an immediate corollary of the next theorem, which shows a closure property of $Sat(\gamma)$. The theorem is due to Eike Best and the author.

Lemma 3.35. *Let N be an occurrence net and let c_1, c_2 be two co-sets of N. Then:*

$$c_1 \cap c_2 \subseteq (c_1 \sqcap c_2) \cap (c_1 \sqcup c_2)$$
$$c_1 \cup c_2 \supseteq (c_1 \sqcap c_2) \cup (c_1 \sqcup c_2)$$

Proof. We prove the first inclusion. The second is analogous. Let $b \in c_1 \cap c_2$. We have that b is not in conflict with c_1 or c_2, because c_1 and c_2 are co-sets. Since $b \in c_2$, $b \not\prec c_2$. Since $b \in c_1$, $b \not\prec c_1$. By the definition of $c_1 \sqcap c_2$ and $c_1 \sqcup c_2$, we have $b \in c_1 \sqcap c_2$ and $b \in c_1 \sqcup c_2$. □

Theorem 3.36 (Closure property of $Sat(\gamma)$). *Let γ be a conjunction of literals, and let C, C' be two elements of $Sat(\gamma)$. If $C \cup C'$ is a configuration, then $C \cup C' \in Sat(\gamma)$.*

Proof. Let $c = Cut(C)$ and $c' = Cut(C')$. The proof is by induction on the structure of γ.

(i) $\gamma =$ **true**. Obvious.

(ii) $\gamma = s$. By the definition of satisfaction, we have to show $s \in p_m(c \sqcup c')$. Since C, $C' \in Sat(s)$, we have:

$$\exists b \in c \; p_m(b) = s \quad \text{and} \quad \exists b' \in c' \; p_m(b') = s$$

If $b = b'$ then $b \in c \cap c'$. By Lemma 3.35, $b \in c \sqcup c'$. So $s \in p_m(c \sqcup c')$. If $b \neq b'$ then, since $p_m(b) = p_m(b')$ and b, b' cannot be in conflict (otherwise their input events are in conflict, which contradicts the fact that $C \cup C'$ is a configuration), either $b \prec b'$ or $b' \prec b$. By the definition of $c \sqcup c'$, in the former case $b' \in c \sqcup c'$. In the latter case, $b \in c \sqcup c'$. Hence in both cases $s \in p_m(c \sqcup c')$.

(iii) $\gamma = \neg s$. Let $c = Cut(C)$ and $c' = Cut(C')$. By the definition of satisfaction, we have to show $s \notin p_m(c \sqcup c')$. Since $C \models \neg s$ and $C' \models \neg s$, we have $s \notin p_m(c)$ and $s \notin p_m(c')$. Since $c \sqcup c' \sqsubseteq c \sqcup c'$ by Lemma 3.35, we have $s \notin p_m(c \sqcup c')$.

(iv) $\gamma = \gamma_1 \wedge \gamma_2$. Then, $C \in Sat(\gamma_1)$ and $C \in Sat(\gamma_2)$, and the same holds for C'.

By induction hypothesis, $C \cup C' \in Sat(\gamma_1)$ and $C \cup C' \in Sat(\gamma_2)$. So $C \cup C' \in Sat(\gamma_1 \wedge \gamma_2)$. □

Corollary 3.37. *Let $C_2 \in \mathcal{F}$ and γ a conjunction of literals. Then, $Last(\emptyset, \gamma, \{C_2\})$ contains at most one configuration.*

Proof. Let C, C' be two configurations of $Last(\emptyset, \gamma, \{C_2\})$. Then C and C' belong to $Sat(\gamma)$. Since $C \subseteq C_2$ and $C' \subseteq C_2$, $C \cup C'$ is a configuration. By Theorem 3.36, $C \cup C' \models \gamma$. Since $C \cup C' \subseteq C_2$, we have $C \cup C' \in Sat(\emptyset, \gamma, \{C_2\})$, which implies $C = C \cup C' = C'$, because both C and C' are maximal. □

The problem of computing $Last(\emptyset, \gamma, \{C_2\})$ was studied in [19] by Bernhard von Stengel and the author. We just adapt the solution of [19] to our notations.

Let γ^+ be the set of places s of Σ such that s is a literal of γ, and let γ^- be the set of places s such that $\neg s$ is a literal of γ. The idea of the algorithm is the following. We start with C_2. If C_2 satisfies γ, then we are done because $Last(\emptyset, \gamma, \{C_2\}) = \{C_2\}$. Otherwise, either $Mark(C_2)$ does not contain some place of γ^+, or it contains some place of γ^-. In both cases, we remove some events from C_2, while ensuring that the result is a configuration lying in the future of the *Last* configuration (loosely speaking, we take care of not removing too many events). This procedure is iterated until a solution is found.

Algorithm 3.38.
Input: The configuration C_2, the sets γ^+ and γ^-.
Output: $Last(\emptyset, \gamma, \{C_2\})$.
begin
  $C := C_2$;
  {invariant: C is a configuration, and $Last(\emptyset, \gamma, \{C_2\}) \ll \{C\}$}
  while $C \not\models \gamma$ **do**
    choose $s \in (\gamma^+ \setminus Mark(C)) \cup (\gamma^- \cap Mark(C))$;
    if $s \in \gamma^+ \setminus Mark(C)$ **then**
      if ${}^\bullet C$ contains no s-labelled condition **then return** $\emptyset$
      else $b :=$ maximal s-labelled condition of ${}^\bullet C$;
        $C := C \setminus \{e \in C \mid b \prec e\}$
      endif
    elseif $s \in \gamma^- \cap Mark(C)$ **then**
      $b :=$ unique s-labelled condition of $Cut(C)$;
      if b has no input events **then return** $\emptyset$
      else $e :=$ unique input event of b;
        $C := C \setminus \{e' \in C \mid e \preceq e'\}$
      endif
    endif
  endwhile
  $\{C \models \gamma$ and $Last(\emptyset, \gamma, \{C_2\}) \ll \{C\}\}$
  return $\{C\}$
end

Let us apply the algorithm to compute $Last(\emptyset, \neg s_3 \wedge s_6, \{e_2, e_3, e_4, e_5, e_6\})$ for the finite prefix of Fig. 5. In this case, $\gamma^+ = \{s_6\}$ and $\gamma^- = \{s_3\}$.

C is initialised to $\{e_2, e_3, e_4, e_5, e_6\}$. So $Mark(C) = \{s_3, s_2\}$. Since C does not satisfy $\neg s_3 \wedge s_6$, we enter the loop. We have $\gamma^+ \setminus Mark(C) = \{s_6\}$ and $\gamma^- \cap Mark(C) = \{s_3\}$. We choose $s = s_6$. The maximal s_6-labelled condition is b_5. So the variable b is assigned b_5. We have $\{e' \in C \mid b_5 \prec e\} = \{e_4, e_5, e_6\}$. Therefore, C is assigned the set $\{e_2, e_3\}$, which finishes the first iteration of the loop.

Now $Mark(C) = \{s_3, s_6\}$, and so C does not satisfy $\neg s_3 \wedge s_6$. So we enter the loop again. We have $\gamma^+ \setminus Mark(C) = \emptyset$ and $\gamma^- \cap Mark(C) = \{s_3\}$, so we choose $s = s_3$. The variable e is assigned the event e_2. Since we have $\{e' \in C \mid e_2 \preceq e'\} = \{e_2\}$, C is set to $\{e_3\}$, which finishes the second iteration of the loop.

Now $Mark(C) = \{s_1, s_6\}$, and so C does satisfy $\neg s_3 \wedge s_6$. $\{\{e_3\}\}$ is returned as result.

The correctness proof can be found in [19]. We summarize the main points. Notice first that the invariant holds before the loop, and that the postcondition implies $Last(\emptyset, \gamma, \{C_2\}) = \{C\}$. Moreover, the loop terminates because every iteration step decreases C.

If $s \in \gamma^+ \setminus Mark(C)$ and the algorithm returns the empty set, then no configuration contained in C satisfies s, because otherwise some condition of ${}^\bullet C$ would be s-labelled. Therefore, no configuration contained in C satisfies γ. Together with the invariant, this implies that no configuration at all satisfies γ, and thus $Last(\emptyset, \gamma, \{C_2\}) = \emptyset$. Similarly, if $s \in \gamma^- \cap Mark(C)$ and the algorithm returns the empty set, then no configuration contained in C satisfies $\neg s$ because, since b has no input events, for every configuration contained in C, b belongs to $Cut(C)$. Again, together with the invariant, this implies that no configuration at all satisfies γ, and thus $Last(\emptyset, \gamma, \{C_2\}) = \emptyset$.

If the algorithm terminates and returns a singleton, then the loop has terminated properly, which implies that both the invariant and the negation of the loop condition hold; the conjunction of both implies $Last(\emptyset, \gamma, \{C_2\}) = \{C\}$.

It remains to show that the loop preserves the invariant. For the first alternative, notice that $C \setminus \{e \in C \mid b \prec e\}$ is the largest configuration contained in C that satisfies $\neg s$. For the second, notice that every configuration between $C \setminus \{e' \in C \mid e \preceq e'\}$ and C satisfies s.

To evaluate the complexity of the algorithm, we first observe that the loop is executed at most $|C_2|$ times. The complexity of computing the sets $C \setminus \{e \in C \mid b \prec e\}$ and $C \setminus \{e' \in C \mid e \preceq e'\}$ depends on the implementation of the causal relation $\prec$. Let $N = (B, C_2, F)$ be the causal net having C_2 as set of events. An optimal implementation in which each node x has pointers to all the nodes of $x^\bullet$ allows to compute both sets in $O(|B|)$ time. However, with such an implementation the construction of the finite prefix becomes more complicated. An easier implementation of $\prec$ is as a list of pairs $(x, x^\bullet)$ for each node $x \in B \cup C_2$. In this case, the time is $O(|B|^2)$. Since $|C_2| \leqslant |B|$, the total running time is $O(|B|^2)$ or $O(|B|^3)$, depending on the implementation.

3.6. The model checker

In the previous sections we have gradually reduced the model checking problem to simpler problems, for which finally we were able to provide algorithms. It is time to recapitulate and describe the global structure of the model checker.

As input of the model checker we need the finite prefix β_f of the system Σ, together with its set $\mathcal{D}$ of maximal configurations, and a formula ϕ. β_f and $\mathcal{D}$ can be computed using McMillan's algorithm (actually, the algorithm of [29] computes only β_f, but it is not difficult to modify it so that it returns $\mathcal{D}$ as well). The model checker makes use of the following three functions:

- `normalform`(ϕ). Accepts a formula ϕ of the logic. Returns a set $\{\phi_1, \ldots \phi_n\}$ of formulas in normal form such that $\phi \equiv \bigvee_{i=1}^{n} \phi_i$.
- $\mu S_f.(\mathcal{C})$. Accepts a set $\mathcal{C} \subseteq \mathcal{F}$. Returns the set $\mu \mathcal{S}_f.\mathcal{C}$.
- `basic-Last`$(\mathcal{C}_1, \gamma, \mathcal{C}_2)$. Accepts two sets of configurations $\mathcal{C}_1, \mathcal{C}_2 \subseteq \mathcal{F}$ and a conjunction of literals γ. Returns the set $Last(\mathcal{C}_1, \gamma, \mathcal{C}_2)$.

These functions can be computed using the results of Sections 4, 6 and 7 respectively.

We define the function `Last`$(\mathcal{C}_1, \phi, \mathcal{C}_2)$, which accepts two sets of configurations $\mathcal{C}_1$, $\mathcal{C}_2 \subseteq \mathcal{F}$ and a formula ϕ in normal form and returns the set $Last(\mathcal{C}_1, \phi, \mathcal{C}_2)$. We write `Last`$(\phi)$ for `Last`$(\emptyset, \phi, \mathcal{D})$.

function `Last`$(\mathcal{C}_1, \phi, \mathcal{C}_2)$
begin
  if $\phi = \gamma$ **then return** `basic-Last`$(\mathcal{C}_1, \gamma, \mathcal{C}_2)$
  elseif $\phi = \psi \wedge \Diamond\rho$ **then return** `Last`$(\mathcal{C}_1, \psi, max\{\mathcal{C}_2 \bigtriangledown \mu S_f.\texttt{Last}(\rho)\})$
  elseif $\phi = \psi \wedge \neg\Diamond\rho$ **then return** `Last`$(max\{\mathcal{C}_1 \cup \mu S_f.\texttt{Last}(\rho)\}, \psi, \mathcal{C}_2)$
  endif
end

Finally, the model checker looks as follows (actually, the model checker only accepts formulas of the form $\Diamond\phi$, but formulas without modalities can be easily checked).

Algorithm 3.39 (*The model checker*).
Input: β_f, $\mathcal{D}$, ϕ
Output: The truth value of $\Sigma \models \Diamond\phi$
begin
  Φ := `normalform`(ϕ);
  result := **false**;
  while *result* = **false and** $\Phi \neq \emptyset$ **do**
    choose $\psi \in \Phi$;
    if `Last`$(\psi) \neq \emptyset$ **then** *result* := **true endif**
  endwhile
  return *result*
end

Observe that we may have $\texttt{Last}(\psi) = \{\emptyset\}$, which means that the largest configuration satisfying ψ is the empty one. In this case, $\texttt{Last}(\psi) \neq \emptyset$

Let us apply this model checker to the 1-safe system of Fig. 3 (whose finite prefix is shown in Fig. 5) and the formula $\Box\Diamond s_2$. In this case, the set $\mathcal{D}$ contains three elements:

$$\mathcal{D} = \{\{e_1, e_2\}, \{e_2, e_3, e_4, e_5\}, \{e_2, e_3, e_4, e_6, e_7\}\}$$

The formula is equivalent to $\neg\Diamond\neg\Diamond s_2$. We apply the model checker to the formula $\Diamond\neg\Diamond s_2$. Hence, $\phi = \neg\Diamond s_2$ and $\Phi := \{\textbf{true} \wedge \neg\Diamond s_2\}$. We have to compute $\texttt{Last}(\textbf{true} \wedge \neg\Diamond s_2)$. According to the definition of the function Last, the first step is the computation of $\texttt{basic-Last}(s_2)$. We get

$$\texttt{basic-Last}(s_2) = \{\{e_1, e_2\}, \{e_2, e_3, e_4, e_5, e_6\}\}$$

and

$$\mu S_f.\,(\texttt{basic-Last}(s_2)) = \mathcal{D}$$

Therefore

$$\texttt{Last}(\textbf{true} \wedge \neg\Diamond s_2) = \texttt{Last}(max\{\mathcal{D} \cup \mathcal{D}\}, \textbf{true}, \mathcal{D}) = \texttt{Last}(\mathcal{D}, \textbf{true}, \mathcal{D})$$

Since no configuration C of β_f satisfies $C \not\ll \mathcal{D}$ we have $\texttt{Last}(\mathcal{D}, \textbf{true}, \mathcal{D}) = \emptyset$. Then, $\Sigma \not\models \Diamond\neg\Diamond s_2$ and, finally, $\Sigma \models \neg\Diamond\neg\Diamond s_2$.

The complexity of the algorithm in the size of the system depends on two parameters: the number of maximal configurations of the finite prefix β_f — that is, the cardinality of the set $\mathcal{D}$ — and the largest size of the configurations of $\mathcal{D}$. Consider a formula with one $\Diamond$-symbol. In this case, the model checker has to solve a *Last* problem for each element of $\mathcal{D}$, and the size of these problems is essentially bounded by the size of the largest maximal configuration.

Of these two parameters, the first is much more critical than the second. If, at some reachable marking, many different conflicts can be concurrently solved, then the finite prefix "splits" at this point into a number of processes proportional to the product of the possible solutions of the conflict. In the next chapter we show that, if the set $\mathcal{D}$ contains one single element, then the model checker performs very well, and for conflict-free systems can solve problems in polynomial time for which the known algorithms require exponential time.

4. Persistent and conflict-free systems

The interplay of concurrency and conflicts greatly complicates the analysis of a system. Concurrency theory has therefore studied with particular interest systems in which two simultaneously enabled actions can always be concurrently executed (i.e., it is never the case that the system can only execute one out of two actions). The persistent systems of [27], the marked graphs or synchronisation graphs of [14, 20],

the conflict-free systems of [25, 27] or the confluent processes of [30] are all examples of models defined to exploit the possibilities of this restriction.

We study the complexity of the model checker for the class of 1-safe persistent systems. The model checker is shown to be polynomial in the size of the finite prefix, and linear in the length of formulas in normal form. We then study the class of conflict-free systems [27]. For this class, the model checker is shown to be polynomial in the size of the system itself, not of its finite prefix, and linear in the size of formulas in normal form. To the best of our knowledge, all verification methods described in the literature require exponential time in the size of the system for the logic L and conflict-free systems.

4.1. Persistent systems

Persistent systems play an important rôle in the verification of asynchronous circuits [35, 37]. As pointed out in [37], 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.

Persistent systems are defined with a notion of concurrency in mind that is slightly more liberal than ours. The consequence is that not always two enabled actions of a persistent system are concurrent in our more restrictive sense. Therefore, we define and study the smaller class of strongly persistent systems. However, we show later in this section that every persistent system can be easily simulated by a strongly persistent one, which allows us to extend our results to persistent systems.

Definition 4.1 (*Concurrent transitions*). Let N be a net, M a marking of N and t_1, t_2 two different transitions enabled at M. The transitions t_1 and t_2 are concurrent at M if for every $s \in {}^\bullet t_1 \cap {}^\bullet t_2$: $M(s) \geqslant 2$.

The following characterisation of concurrency in 1-safe systems is well known:

Proposition 4.2 (Concurrent transitions in 1-safe systems). *Let Σ be a 1-safe system, M a reachable marking of Σ and t_1, t_2 two different transitions enabled at M_0. The transitions t_1 and t_2 are concurrent at M iff* $({}^\bullet t_1 \cup t_1^\bullet) \cap ({}^\bullet t_2 \cup t_2^\bullet) = \emptyset$.

Proof. ($\Rightarrow$): Since every place contains at most one token at M, we have ${}^\bullet t_1 \cap {}^\bullet t_2 = \emptyset$. Also, $t_1^\bullet \cap t_2^\bullet = \emptyset$, because otherwise $M \xrightarrow{t_1 t_2} M'$ is an occurrence sequence and M' a marking that puts two tokens in the places of $t_1^\bullet \cap t_2^\bullet$, which contradicts the 1-safeness of the system. The rest of the statement is proved similarly.

($\Leftarrow$): Trivial. □

Definition 4.3 (*Strongly persistent systems*). A system Σ is strongly persistent if for every reachable marking M and every pair t_1, t_2 of different transitions of N enabled at M, the transitions t_1, t_2 are concurrent at M.

We can now characterise strongly persistent systems in terms of their maximal branching processes.

Theorem 4.4 (Characterisation of strongly persistent systems). *Let Σ be a 1-safe system. Σ is strongly persistent iff its maximal branching process is a process.*

Proof. ($\Rightarrow$): Assume $\Sigma = (S, T, F, M_0)$ is not strongly persistent. Then, there exists a reachable marking M, two different transitions t_1, $t_2 \in T$ enabled at M and a place $s \in {}^\bullet t_1 \cap {}^\bullet t_2$ such that $M(s) = 1$.

Let c be a cut of β_m such that $p_m(c) = M$ and $|p_m(c)| = |M|$. In particular, c contains a unique condition b such that $p(b) = s$. Since M enables t_1 and t_2 and Σ is 1-safe, there exist two subsets c_1, $c_2 \subseteq c$ such that $p_m(c_1) = {}^\bullet t_1$ and $p_m(c_2) = {}^\bullet t_2$.

Let (N, p_m) be the prefix of β_m having as nodes the predecessors of the conditions of c. By the definition of β_m, (N, p_m) can be extended to another prefix (N', p_m) in the following way:

- Add two new events e_1, e_2 to N satisfying ${}^\bullet e_i = c_i$ and $p_m(e_i) = t_i$ for $i = 1, 2$.
- For every place $s \in t_i^\bullet$, $i = 1, 2$, add a new condition b to N satisfying ${}^\bullet b = \{e_i\}$ and $p_m(b) = s$.

By the definition of N', $b \in {}^\bullet e_1 \cap {}^\bullet e_2$. So e_1, e_2 are in conflict in N'.

Since (N', p_m) is a prefix of β_m, β_m contains these two events in conflict as well. Therefore, the net N_m underlying β_m is not a causal net, which implies that β_m is not a process.

($\Leftarrow$): We show that every finite prefix β of β_m is a process. Since every condition of β_m is finitely preceded, this proves that no node of β_m has more than one output event, which implies the result.

The proof is by induction on the set of events E of β (notice that a prefix of β_m is characterised by its set of events).

Base. $E = \emptyset$. Obvious.

Step. β is a process, prefix of β_m, with E as set of events.

Let β' be a prefix of β_m having $E \cup \{e'\}$ ($e' \notin E$) as set of events, and assume that β' is not a process. Then $e \# e'$ for some event $e \in E$. Moreover, we can assume ${}^\bullet e \cap {}^\bullet e' \neq \emptyset$ (where the presets refer to β').

We claim that $B_0 = {}^\bullet e \cap {}^\bullet e'$ is a nonempty co-set. Notice that, since B_0 is a set of conditions of β, and β is a process, no two conditions of B_0 are in conflict. Therefore, it suffices to prove that for every b, $b' \in B_0$: $\neg(b \prec b')$. Assume $b \prec b'$ for some b, $b' \in B_0$. If b, $b' \in {}^\bullet e'$ then e' is in self-conflict, contradicting the hypothesis that β' is a branching process. Symmetrically, if b, $b' \in {}^\bullet e$ then e is in self-conflict. If $b \in {}^\bullet e$ and $b' \in {}^\bullet e'$, then — since e and e' are in conflict — e and b' are in conflict. Since both e and b' are nodes of β, this contradicts the hypothesis that β is a process. Since we reach a contradiction in all cases, the claim is proved.

Now, let c be a cut of β' containing B_0. Then, $p_m(c)$ is a reachable marking of Σ. By the definition of B_0, ${}^\bullet p_m(e) \cup {}^\bullet p_m(e') \subseteq p_m(c)$. So both $p_m(e)$ and $p_m(e')$ are enabled at $p_m(c)$. Since Σ is 1-safe, $p_m(e) \neq p_m(e')$. Finally, since there exists $b \in {}^\bullet e \cap {}^\bullet e'$,

we have $p_m(b) \in {}^\bullet p_m(e) \cap {}^\bullet p_m(e')$. So $p_m(e)$ and $p_m(e')$ are both enabled but not concurrent at $p_m(c)$, which contradicts the hypothesis that Σ is strongly persistent. □

The main property of strongly persistent systems in connection with our model checker, which we are going to prove now, is that for every formula ϕ, the set $Last(\phi)$ always contains at most one configuration.

The result is obtained using as lemma a much stronger version of the closure property for *Sat* sets that we obtained in Section 3 (Theorem 3.36). We showed there that for every conjunction of literals γ and every two configurations C, C' of $Sat(\phi)$, if $C \cup C'$ is a configuration then $C \cup C'$ also belongs to $Sat(\gamma)$. We strengthen this result in two ways:

- Since the maximal branching process is a process, its set of finite configurations is a complete lattice; therefore, for every two configurations C, C', the set $C \cup C'$ is a configuration as well, which implies that the condition on $C \cup C'$ can be suppressed.
- The result is generalised from conjunctions of literals to all formulas in normal form.

Theorem 4.5 (Closure property of $Sat(\phi)$ for strongly persistent systems). *Let Σ be a strongly persistent 1-safe system and let ϕ be a formula in normal form. Let C, C' be two finite configurations of the maximal branching process of Σ that belong in $Sat(\phi)$. Then $C \cup C' \in Sat(\phi)$.*

Proof. By Theorem 4.4, the maximal branching process of Σ is a process. Therefore, its set of configurations is a complete lattice, with union as least upper bound, which implies that $C \cup C'$ is a finite configuration.

The proof is by induction on the structure of ϕ.

(i) $\phi = \gamma$. Follows from Theorem 3.36.

(ii) $\phi = \psi \wedge \Diamond\rho$. We have:

- C, $C' \in Sat(\psi)$, and
- there exist $C_1 \supseteq C$, $C_1' \supseteq C'$ such that C_1, $C_1' \in Sat(\rho)$.

Since both $C \cup C'$ and $C_1 \cup C_1'$ are finite configurations, by induction hypothesis $C \cup C' \in Sat(\psi)$ and $C_1 \cup C_1' \in Sat(\rho)$. Since $C_1 \cup C_1' \supseteq C \cup C'$, we get $C \cup C' \in Sat(\psi \wedge \Diamond\rho)$.

(iii) $\phi = \psi \wedge \neg\Diamond\rho$. We have:

- C, $C' \in Sat(\psi)$, and
- for every $C_1 \supseteq C$ and $C_1' \supseteq C'$, $C_1 \notin Sat(\rho)$ and $C_1' \notin Sat(\rho)$.

By induction hypothesis, $C \cup C' \in Sat(\psi)$. Clearly, for every configuration $C'' \supseteq C \cup C'$, $C'' \notin Sat(\rho)$. So $C \cup C' \in Sat(\psi \wedge \neg\Diamond\rho)$. □

It is easy to see that the theorem is not true for arbitrary formulas. Consider the process of Fig. 6 and the formula $s_1 \vee s_2$, which is not in normal form. The configurations $\{e_1\}$ and $\{e_2\}$ satisfy $s_1 \vee s_2$; however, the configuration $\{e_1, e_2\}$ does not.

For a formula ϕ, define $l(\phi)$ as the number of $\Diamond$-operators of ϕ plus 1. The problem of computing $Last(\emptyset, \gamma, \{C\})$ for a configuration C and a conjunction of literals γ is called a *Last-problem*.

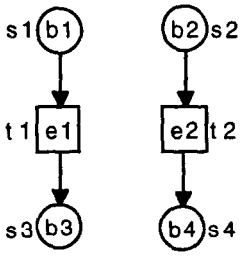

Fig. 6. A process.

Theorem 4.6 (Complexity of the model checker for strongly persistent systems). *Let ϕ be a formula in normal form.*

(1) *$Last(\phi)$ has at most one element, and*

(2) *$Last(\phi)$ can be computed solving $l(\phi)$ Last-problems.*

Proof. (1) Assume $Last(\phi)$ contains two different elements C_1, C_2. By Theorem 4.5, $C_1 \cup C_2 \in Sat(\phi)$, which contradicts the maximality of C_1, C_2.

(2) We prove a more general result by induction on the structure of ϕ. If C_1, C_2 contain at most one element, then $Last(C_1, \phi, C_2)$ can be computed by solving $l(\phi)$ *Last*-problems.

(i) $\phi = \gamma$. Then $l(\phi) = 1$, and the result follows.

(ii) $\phi = \psi \wedge \Diamond\rho$. By the compositional equations for *Last*, we have

$$Last(C_1, \psi \wedge \Diamond\rho, C_2) = Last(C_1, \psi, max\{C_2 \bigtriangledown C\})$$

where $C = \mu\mathcal{S}_f.Last(\rho)$. We claim that $max\{C\}$ contains at most one element.

$$\begin{aligned}
&max\{\mu\mathcal{S}_f.Last(\rho)\}\\
&= \{\text{definitions of } \mu\mathcal{S}_f \text{ and } Last\}\\
&\quad max\{\mu\mathcal{S}.max\{Sat_f(\rho)\} \bigtriangledown \{E_f\}\}\\
&= \{\text{monotonicity of } \mathcal{S}_e, \text{ properties of } max\}\\
&\quad max\{\mu\mathcal{S}.Sat_f(\rho) \bigtriangledown \{E_f\}\}\\
&= \{\text{ Theorem 3.18}\}\\
&\quad max\{Sat(\rho)\} \bigtriangledown \{E_f\}
\end{aligned}$$

Since $Sat(\rho)$ is closed under union by Theorem 4.5, $max\{Sat(\rho)\}$ has at most one element. So $max\{Sat(\rho)\} \bigtriangledown \{E_f\}$ has at most one element, and the claim is proved.

By induction hypothesis, $Last(\rho)$ and $Last(C_1, \psi, max\{C_2 \bigtriangledown C\})$ can be computed by solving $l(\rho)$ and $l(\psi)$ *Last*-problems, respectively. By the claim, $Last(C_1, \psi \wedge \Diamond\rho, C_2)$ can be computed by solving $l(\psi) + l(\rho)$ problems. Since the number of $\Diamond$-operators in $\psi \wedge \Diamond\rho$ is $(l(\psi) - 1) + (l(\rho) - 1) + 1$, we have $l(\psi \wedge \Diamond\rho) = l(\psi) + l(\rho)$, which proves the result.

(iii) $\phi = \psi \wedge \neg\Diamond\rho$. Similar to (ii). □

Recall that *Last*-problems can be solved in polynomial time in the size of the finite prefix using the second algorithm presented in Section 3.7. Therefore, for strongly persistent systems our model checker has linear complexity in the length of the formula (in normal form) and polynomial complexity in the size of the finite prefix. To finish our study, we relate the size of the finite prefix to the size of the system. We need the following lemma, which is valid for all 1-safe systems and relates the number of nodes of a process to the length of its longest line. A *line* of a branching process is a maximal set of nodes which are pairwise ordered by the causal relation.

Lemma 4.7 (Number of nodes of a process). *Let* $\pi = (B, E, F, p)$ *be a process of a* 1*-safe system* $\Sigma = (S, T, F, M_0)$ *such that no line of* π *has length greater than* k*. Then* $|B \cup E| \leqslant k \cdot |S|$.

The lemma has an intuitive proof: a process can be depicted within a rectangle, whose longest side corresponds to the direction of the unfolding. The number of nodes of the process is proportional to the area of this rectangle. The width of the rectangle is the maximal number of conditions of a cut; in a 1-safe system, this number is at most the cardinality of $|S|$, because no two conditions of a cut may have the same label. The length of the rectangle is the maximal number of nodes of a line. So the area is the product of both.

Proof. In this proof, a "chain of π" is a chain of nodes of π with respect to the causal relation.

Define the mapping $d: B \cup E \to \mathbb{N}$ as follows:

$$\forall x \in B \cup E: d(x) = \text{maximal length of the chains of } \pi \text{ having } x \text{ as last node}$$

Clearly, $d(x) \leqslant k$ for every $x \in B \cup E$. It follows that

$$|B \cup E| \leqslant \sum_{i=1}^{k} |\{x \in B \cup E \mid d(x) = i\}|$$

We show that for every i, $1 \leqslant i \leqslant k$, $|\{x \in B \cup E \mid d(x) = i\}| \leqslant |S|$, which proves the result. Let x, x' be two different nodes of $B \cup E$ such that $d(x) = d(x')$. We claim that x and x' are not causally ordered. Assume without loss of generality $x \preceq x'$. Let h be a chain of π having x as last node and length $d(x)$. Since $x \preceq x'$, the chain h can be extended to a longer chain having x' as last node, which contradicts $d(x') = d(x)$ and proves the claim. Replace every event of the set $\{x \in B \cup E \mid d(x) = i\}$ by one of its output conditions. Clearly, the new set so constructed contains only pairwise unordered conditions. So it is a co-set and has the same cardinality than $\{x \in B \cup E \mid d(x) = i\}$. By the 1-safeness of Σ, the cardinality of the new set is less or equal to $|S|$, and we are done. □

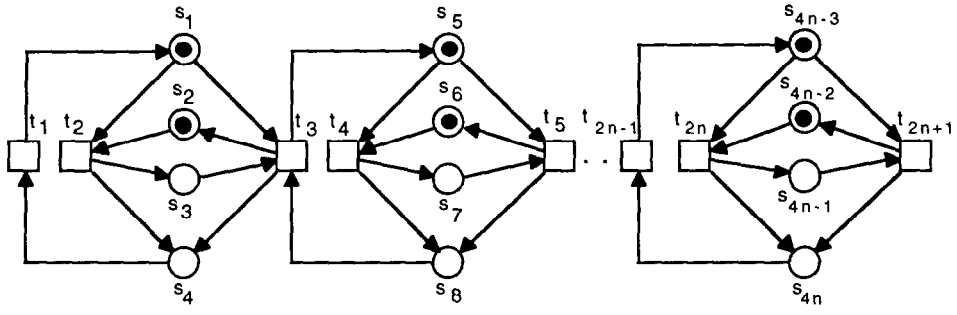

Fig. 7. A family of 1-safe strongly persistent systems with large finite prefix.

Lemma 4.8 (Size of the finite prefix of a strongly persistent system). *Let Σ be a 1-safe strongly persistent system, and let S be the set of places of Σ. The number of nodes of its finite prefix is bounded by $2^{|S|} \cdot |S|$.*

Proof. We show that a line of the finite prefix can contain at most as many events as the number of reachable markings of Σ. Assume there exist a line containing at least n events, where n is the number of reachable markings of Σ. Let e be the nth event of the line. Then, there exist two events $e_1 \prec e_2 \preceq e$ of the line such that $Mark(e_1) = Mark(e_2)$, which implies that e_2 is a cut-off event. By the definition of the finite prefix, e_2 is the last event of the line, and therefore $e_2 = e$. So the line contains exactly n events. Since Σ is 1-safe, the number of reachable markings of Σ is bounded by $2^{|S|}$. The result follows from Lemma 4.7. □

In order to show that this upper bound cannot be substantially improved, we consider the family of net systems of Fig. 7 [15]. The members of the family are 1-safe and strongly persistent; moreover, their finite prefixes contain $\Omega(2^{|S|})$ nodes. To convince ourselves of this, we observe that the marking

$$\{s_1, s_3, s_5, s_7, \ldots, s_{4n-3}, s_{4n-1}\}$$

is reachable, and that it can only be reached from the initial marking after the occurrence of an exponential number of transitions. The reason is that the transition t_{2n-1} must occur at least one, and for every $1 \leqslant i \leqslant n$, the number of occurrences of t_{2i-1} in any occurrence sequence is at least twice the number of occurrences of t_{2i+1}. Since the marking is represented in the finite prefix, all the events corresponding to these occurrences must be contained in the finite prefix.

As mentioned at the beginning of the section, strongly persistent net systems are a subclass of the slightly larger and well known class of persistent systems [27]. In persistent systems, if two transitions are enabled at a marking then they can occur in any order, but not always concurrently.

Definition 4.9 (*Persistent systems*). A net system is persistent if whenever a marking enables two transitions t_1 and t_2, it enables the sequences $t_1 t_2$ and $t_2 t_1$.

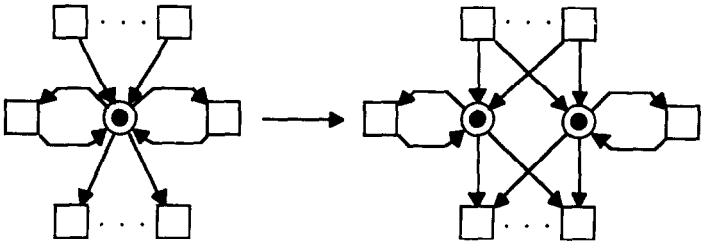

Fig. 8. Transforming a persistent net system into a strongly persistent one.

Strong persistence implies persistence, but the contrary does not hold. Persistent but not strongly persistent 1-safe systems always contain a place s and two different transitions t_1, t_2 such that $\{t_1, t_2\} \subseteq {}^\bullet s \cap s^\bullet$. At some reachable marking M, both t_1 and t_2 are enabled, but cannot happen concurrently because of the place s. s is called a *multiple self-loop* place.

Persistent systems can be simulated by strongly persistent systems. The simulating strongly persistent system can be obtained from the persistent one by means of a simple transformation which removes multiple self-loop places. The transformation is graphically illustrated in Fig. 8. Loosely speaking, every multiple self-loop place is split into several places, as shown in the figure.

It is routine to prove the following: a persistent 1-safe system Σ satisfies a formula ϕ if and only if the strongly persistent system that simulates Σ satisfies a formula $\tilde{\phi}$, where $\tilde{\phi}$ is obtained by substituting every occurrence in ϕ of a self-loop place s by any of the places into which it is split.

4.2. *Conflict-free systems*

As we have seen, the model checker for strongly persistent systems may require exponential time because the finite prefix may be exponential in the size of the system. In this section we study the 1-safe systems in which every place has at most one output transition, which we call 1-out systems. It follows easily from the definition that 1-out systems are strongly persistent. We prove that, for this particular subclass, the finite prefix is cubic in the size of the system — the proof is a slight generalisation of a result of [41]. It is easy to prove by inspection of McMillan's algorithm that for these systems not only the finite prefix has cubic size but can also be constructed in cubic time in the size of the system. Finally, we show that the results for 1-out systems can be easily extended to the class of conflict-free systems.

Definition 4.10 (*1-out systems*). A net system is 1-out if every place has at most one output transition.

Fig. 9 displays again the family of systems we considered in the introduction to this paper as models of concurrent buffers. We now observe that it is a family of 1-out 1-safe systems.

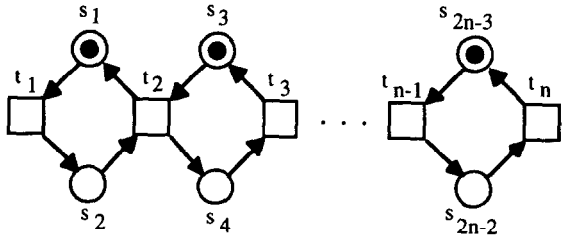

Fig. 9. A simplified model of a concurrent buffer of capacity $n-1$.

The polynomiality of the finite prefix is proved in two steps. First, we show using results of [15,41] that every occurrence sequence of a 1-safe 1-out system of length greater than

$$\frac{|T| \cdot (|T|+1)}{2}$$

(where T is the set of transitions of the system) can be reorganised into another occurrence sequence that visits twice the same marking (Theorem 4.13). Then, we use this result to obtain an upper bound on the maximal length of the lines of the finite prefix; finally, using the lemma that relates this parameter to the size of the finite prefix we obtain a bound on the size of the finite prefix (Theorem 4.14).

Notation 4.11 (*Alphabet and Parikh mapping of a sequence of transitions*). Let Σ be a system and let σ be a sequence of transitions of Σ. The alphabet of σ is the set of transitions that occur in it; we denote it by $\alpha(\sigma)$. For a transition t, $\mathcal{P}(\sigma)(t)$ denotes the number of occurrences of t in σ. $\mathcal{P}(\sigma)$ is called the Parikh mapping of σ.

The following lemma is a reformulation of Lemma 3.4 in [15]. It shows that an occurrence sequence σ can be reorganised as a concatenation of sequences; the first one contains one single occurrence of all the transitions that occur at least once in σ; the second one contains one single occurrence of all the transitions that occur at least twice in σ, and so on.

Lemma 4.12 ([15]). *Let $\Sigma = (S,T,F,M_0)$ be a 1-safe 1-out system, and let σ be an occurrence sequence. Then, there exists an occurrence sequence $\sigma_1\sigma_2\ldots\sigma_k$ such that:*

(1) *$\sigma_1\sigma_2\ldots\sigma_k$ is a permutation of σ.*

(2) *For every transition t, for every i, $1 \leqslant i \leqslant k$: $\mathcal{P}(\sigma_i)(t) \leqslant 1$, and*

(3) *For every i, $1 \leqslant i \leqslant (k-1)$: $\alpha(\sigma_i) \supseteq \alpha(\sigma_{i+1})$.*

We can now prove the following theorem (slightly stronger than Theorem 4.3 in [15]).

Theorem 4.13. *Let $\Sigma = (S,T,F,M_0)$ be a 1-safe 1-out system, and let σ be an occurrence sequence such that $|\sigma| > (|T| \cdot (|T|+1))/2$. Then, there exists an occurrence sequence $\sigma_1\sigma_2\sigma_3$, which is a permutation of σ, where σ_2 is nonempty and*

$$M_0 \xrightarrow{\sigma_1} M_1 \xrightarrow{\sigma_2} M_1 \xrightarrow{\sigma_3} M$$

Proof. By Lemma 4.12, there exists an occurrence sequence $\tau_1 \ldots \tau_k$ satisfying conditions (1)–(3). By (2), $|\tau_1| \leqslant |T|$. By (2) and (3), $|\tau_i| \geqslant |\tau_{i+1}|$ for every i, $1 \leqslant i \leqslant n-1$.

Assume that $|\tau_i| > |\tau_{i+1}|$ for every i, $1 \leqslant i \leqslant n-1$. Then $k \leqslant |T|$ and we have

$$|\sigma| = \sum_{i=1}^{k} |\tau_i| \leqslant \sum_{j=0}^{|T|-j} |T| - j = \frac{|T| \cdot (|T|+1)}{2}$$

Therefore, by our hypothesis, there exists an i such that $|\tau_i| = |\tau_{i+1}|$. We show that taking $\sigma_1 = \tau_1 \ldots \tau_{i-1}$, $\sigma_2 = \tau_i$ and $\sigma_3 = \tau_{i+1} \ldots \tau_k$ the result follows.

Let $M_0 \xrightarrow{\tau_1 \ldots \tau_{i-1}} M_1 \xrightarrow{\tau_i} M_2 \xrightarrow{\tau_{i+1}} M_3$. Since $|\tau_i| = |\tau_{i+1}|$, we have by (3) $\mathcal{P}(\tau_i) = \mathcal{P}(\tau_{i+1})$. By the occurrence rule $M_1 - M_2 = M_2 - M_3$. Since M_1, M_2 and M_3 are 1-safe markings, $M_1 = M_2 = M_3$. So $M_1 \xrightarrow{\tau_i} M_1$, which proves the result. □

Theorem 4.14 (Bound on the size of the finite prefix of a 1-out system). *Let* $\Sigma = (S, T, F, M_0)$ *be a* 1*-safe* 1*-out system. The number of nodes of the finite prefix of* Σ *is* $O(|S| \cdot |T|^2)$.

Proof. By Lemma 4.7, it suffices to show that all lines of the finite prefix have length $O(|T|^2)$. Let h be a chain of the finite prefix of Σ containing $\frac{1}{2}(|T| \cdot (|T|+1)) + 1$ events. We show that h is a line.

Let e be the maximal event of h. Then, all events of h belong to $[e]$, which implies $|[e]| > \frac{1}{2}(|T| \cdot (|T|+1))$. Let σ be the image under p_m of a linearisation of $[e]$. σ is an occurrence sequence of Σ. Then, there exists an occurrence sequence $\sigma_1\sigma_2\sigma_3$, which is a permutation of σ, satisfying the conditions of Theorem 4.13. There exist two configurations C_1, $C_2 \subseteq [e]$ such that σ_1 and $\sigma_1\sigma_2$ are linearisations of C_1 and C_2, respectively. So $C_1 \subset C_2$ and $Mark(C_1) = Mark(C_2)$. Since C_1 and C_2 correspond to the same reachable marking, $\Uparrow Cut(C_1)$ is isomorphic to $\Uparrow Cut(C_2)$.

Let I be an isomorphism from $\Uparrow Cut(C_1)$ to $\Uparrow Cut(C_2)$. Consider the event $I^{-1}(e)$. We have $p_m(I^{-1}(e)) = p_m(e)$ and, by the 1-safeness of Σ, $I^{-1}(e) \prec e$. Moreover, we have $[I^{-1}(e)] = C_1 \cup I^{-1}([e] \setminus C_2)$, which implies $Mark([e]) = Mark([I^{-1}(e)])$. So e is a cut-off event, and therefore no successor of e is contained in the finite prefix. It follows that h is a line of the finite prefix. So every line of the finite prefix contains at most $\frac{1}{2}(|T| \cdot (|T|+1)) + 1$ events, and therefore every line of the finite prefix has at most length $(|T| \cdot (|T|+1)) + 3$. □

If we take $n = |S| \cdot |T|$ as the size of the system (S, T, F, M_0), and further assume $|S| \in O(|T|)$, which is a reasonable assumption, we get that the size of the finite prefix is $O(n^{3/2})$. Since a *Last*-problem requires polynomial time in the size of finite prefix, by Theorem 4.6 our model checker is polynomial in the size of the system and linear in the length of a formula in normal form.

In [15], a result similar to the one of Theorem 4.13 has been obtained for live and safe free choice nets. We conjecture that it can be used to give a tight bound on the size of the processes of the finite prefix for the systems of this class.

1-out systems are a subclass of conflict-free systems.

Definition 4.15 (*Conflict-free systems*). A net system Σ is conflict-free if for every place s of Σ, $|s^\bullet| \leqslant 1 \vee s^\bullet \subseteq {}^\bullet s$.

The name "conflict-free" is somewhat unfortunate in our context, because the branching processes of conflict-free systems can contain events in conflict.

The complexity of various verification problems for conflict-free systems has been studied by Howell, Rosier and Yen in several papers [25,26,41]. For general conflict-free systems (not necessarily 1-safe), the model checking problem for a temporal logic defined in [25] is undecidable, whereas the reachability problem is NP-complete. For 1-safe conflict-free systems, the coverability problem and the concurrency problem (the problem of deciding if two given transitions can be simultaneously enabled) were known to be polynomial [17,41].

We can easily extend our results on 1-safe 1-out systems to 1-safe conflict-free systems. It suffices to observe that 1-out systems can simulate conflict-free systems in the same way that strongly persistent systems simulate persistent ones. In particular, both the reachability and the concurrency problems are instances of the model checking problem for our logic, and the formulas that express them are in normal form. Therefore, the quoted results of [17,41] follow as corollaries of our more general theory.

5. Conclusions

We have presented a model checker for the logic L that works on a net unfolding, a partial order semantics of Petri nets developed by Best, Devillers, Engelfriet, Nielsen, Starke, Winskel and others. The model checker belongs to what can be called the classical model checking paradigm: the system is assigned a semantical object, and all the properties of a logic are tested on it (as opposed to the approach based on "on the fly" verification, where a semantical object is computed for each property). While "on the fly" verification may be very efficient for some problems, it does not produce a reusable object which can be stored to check new properties. In this sense, this paper complements the work of Godefroid, Wolper, and others [21,22].

The transition system or Kripke structure which is at the basis of interleaving model checkers is replaced in our approach by the finite prefix of the maximal branching process defined by McMillan [29]. Only local states are directly represented in the prefix; the global states are encoded or embedded in it. In this sense, net unfoldings are related to BDD's (binary decision diagrams). Their main advantage with respect to BDD's is the existence of a mature theory of net unfoldings as partial order semantics of concurrent systems. This theory has lead us to an efficient algorithm for the retrieval of

information from the unfolding. The algorithm computes certain maximal configurations of the processes of the finite prefix which, essentially, can be seen as least upper bounds in lattices. The combination "unfolding + lattice algorithms" seems to be an interesting alternative to the usual combination "state space + traversing algorithms", and we think that it deserves to be further studied.

The verification algorithm for L can check several interesting safety properties, such as the reachability of a marking, coverability problems and liveness of transitions. It has been shown that the algorithm is polynomial in the size of the system for the class of (1-safe) conflict-free systems. This improves previous results, since the known algorithms are either not applicable to the properties expressible in our logic or require exponential time. It also extends the class of properties of conflict-free systems decidable in polynomial time. Conflict-free systems allow (admittedly restrictive) communication between subsystems via merging of transitions. This gives our polynomiality result an special flavour, because such results are usually claimed for systems composed of independent, non-communicating subsystems (as, for instance, in the decomposition result of Groote and Moller [24]).

The extension of our technique to a more expressive logic containing next operators or until operators is an open problem. In particular, our approach depends on the existence of adequate normal forms which do not seem to exist for these logics. Another open problem is the extension of our technique to linear temporal logics. We have obtained some preliminary results using another definition of unfolding, in which we construct a transition system whose nodes are reachable markings and whose transitions are labeled with processes [18]. Further open problems are the extension of the model checker to n-safe systems, and the investigation of the size of the finite prefix for different classes of nets.

Acknowledgments

Very special thanks to Thomas Thielke, who read the paper very carefully and corrected many mistakes. Thanks to Eike Best, Allan Cheng, Jens Palsberg, Kenneth McMillan, Bernhard von Stengel and two anonymous referees for very helpful discussions and suggestions.

References

[1] H.R. Andersen and G. Winskel, Compositional checking of satisfaction, in: K.G. Larsen and A. Skou, eds., *Computer Aided Verification*, Lecture Notes in Computer Science **575** (Springer, Berlin, 1991) 24–36.

[2] M. Ben-Ari, Z. Manna and A. Pnueli, The temporal logic of branching time, *Acta Inform.* **20**(3) (1983) 207–226.

[3] L. Bernardinello and F. De Cindio, A survey of basic net models and modular net classes, in: G. Rozenberg, ed., *Advances in Petri Nets 1992*, Lecture Notes in Computer Science **609** (Springer, Berlin, 1992) 304–351.

[4] E. Best and R. Devillers, Sequential and concurrent behaviour in Petri net theory, *Theoret. Comput. Sci.* **55**(1) (1987) 299–323.

[5] E. Best and J. Esparza, Model checking of persistent Petri nets, in: E. Börger, G. Jäger, H. Kleine Büning and M.M. Richter, eds., *Computer Science Logic 91*, Lecture Notes in Computer Science **626** (Springer, Berlin, 1991) 35–52.

[6] E. Best and C. Fernández, *Nonsequential Processes—A Petri Net View*, EATCS Monographs on Theoretical Computer Science, Vol. 13 (Springer, Heidelberg, 1988).

[7] J.C. Bradfield, Verifying temporal properties of systems with applications to Petri nets, Ph.D. Thesis, University of Edinburgh (1991).

[8] M. Browne, E.M. Clarke and D.L. Dill, Automatic circuit verification using temporal logic: Two new examples, in: *IEEE International Conference on Computer Design: VLSI and Computers* (1985).

[9] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill and L.J. Hwang, Symbolic model checking: 10^{20} states and beyond, in: *Proceedings of the 5th Annual Symposium on Logic in Computer Science*, Philadelphia, PA (1990) 428–439.

[10] A. Cheng, J. Esparza and J. Palsberg, Complexity results for 1-safe Petri nets, in: *Proceedings of the 13th Conference on Foundations of Software Technology and Theoretical Computer Science*, Bombay (1993).

[11] E.M. Clarke, E.A. Emerson and A.P. Sistla, Automatic verification of finite-state concurrent systems using temporal logic specifications, *ACM Trans. Programming Languages Systems* **8**(2) (1986) 244–263.

[12] E.M. Clarke, O. Grümberg and M.C. Browne, Reasoning about networks with many identical finite-state processes, *Inform. and Comput.* **81** (1989) 13–31.

[13] R. Cleaveland, J. Parrow and B. Steffen, A semantics based verification tool for finite-state systems, in: *Proceedings of the 9th Symposium on Protocol Specification, Testing and Verification* (North-Holland, Amsterdam, 1989).

[14] F. Commoner, A.W. Holt, S. Even and A. Pnueli, Marked directed graphs, *J. Comput. System Sci.* **5** (1971) 511–523.

[15] J. Desel and J. Esparza, Shortest paths in reachability graphs, in: M. Ajmone Marsan, ed., *Proceedings of the 14th Conference on Application and Theory of Petri Nets*, Lecture Notes in Computer Science **691** (Springer, Berlin, 1993) 224–241.

[16] J. Engelfriet, Branching processes of Petri nets, *Acta Inform.* **28** (1991) 575–591.

[17] J. Esparza, A solution to the covering problem for 1-bounded conflict-free Petri nets using Linear Programming, *Inform. Processing Lett.* **41** (1992) 313–319.

[18] J. Esparza, A partial order approach to model checking, Habilitation Thesis, University of Hildesheim (1993).

[19] J. Esparza and B. von Stengel, The asynchronous committee meeting problem, in: *Proceedings of Graph Theoretic Concepts in Computer Science 93* (1993).

[20] H.J. Genrich and K. Lautenbach, Synchronisationsgraphen, *Acta Inform.* **2** (1973) 143–161.

[21] P. Godefroid, Using partial orders to improve automatic verification methods, in: *Proceedings of the Workshop on Computer Aided Verification*, Rutgers Univ., New Brunswick, NJ (1990).

[22] P. Godefroid and P. Wolper, Using partial orders for the efficient verification of deadlock freedom and safety properties, in: K.G. Larsena and A. Skou, eds., *Computer Aided Verification*, Lecture Notes in Computer Science **575** (Springer, Berlin, 1991) 332–343.

[23] D. Gomm, E. Kindler, B. Paech and R. Walter, Compositional liveness properties of EN-systems, in: *Proceedings of the 14th International Conference on Applications and Theory of Petri Nets*, Lecture Notes in Computer Science **691** (Springer, Berlin, 1993) 262–281.

[24] J.F. Groote and F. Moller, Verification of parallel systems via decomposition, LFCS Report 92-193, University of Edinburgh (1992).

[25] R.R. Howell and L. Rosier, On questions of fairness and temporal logic for conflict-free Petri nets, in: G. Rozenberg, ed., *Advances in Petri Nets 1988*, Lecture Notes in Computer Science **340** (Springer, Berlin, 1988) 200–220.

[26] R.R. Howell and L. Rosier, An $O(n^{1.5})$ algorithm to decide boundedness for conflict-free vector replacement systems, *Inform. Processing Lett.* **25**(1) (1987) 27–33.

[27] L.H. Landweber and E.L. Robertson, Properties of conflict-free and persistent Petri nets, *J. ACM* **25**(3) 91978) 352–364.

[28] O. Lichtenstein and A. Pnueli, Checking that finite state programs satisfy their linear specification, in: *Proceedings of the 12th ACM Symposium on Principles of Programming Languages*, New Orleans, LA (1985) 97–107.

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

[30] R. Milner, *Communication and Concurrency* (Prentice-Hall, Englewood Cliffs, NJ, 1989).

[31] M. Nielsen, G. Plotkin and G. Winskel: Petri nets, event structures and domains, *Theoret. Comput. Sci.* **13**(1) (1980) 85–108.

[32] D.K. Probst and H.F. Li, Partial-order model checking, a guide for the perplexed, K.G. Larsen and A. Skou, eds., *Computer Aided Verification*, Lecture Notes in Computer Science **575** (1991) 322–332.

[33] J.P. Quielle and J. Sifakis, Specification and verification of concurrent systems in CESAR, in: *Proceedings of the 5th International Symposium on Programming*, Lecture Notes in Computer Science **137** (Springer, Berlin, 1981) 337–351.

[34] W. Reisig, Concurrent temporal logic, SFB-Bericht Nr. 342/7/91 B, Technische Universität München (1991).

[35] L.Ya. Rosenblum and A.V. Yakovlev, Signal graphs: from self-timed to timed ones, in: *Proceedings of the International Workshop on Timed Petri Nets*, Torino, Italy (IEEE Computer Society, Silver Spring, MD, 1985) 199–207.

[36] P. Starke, Processes in Petri nets, *J. of Inform. Processing and Cybernetics (EIK)* **17** (1981) 389–416.

[37] M. Tiusanen, Some unsolved problems in modelling self-timed circuits using Petri nets, *EATCS Bull.* **36** (1988) 152–160.

[38] A. Valmari, Stubborn sets for reduced state space generation, in: G. Rozenberg, ed., *Advances in Petri Nets 1990*, Lecture Notes in Computer Science **483** (Springer, Berlin, 1990) 491–515.

[39] A. Valmari and M. Clegg, Reduced labelled transition systems save verification effort, in: J.C.M. Baeten and J.F. Groote, eds., *CONCUR'91*, Lecture Notes in Computer Science **527** (Springer, Berlin, 1991) 526–540.

[40] D.J. Walker, Automated analysis of mutual exclusion algorithms using CCS, *Formal Aspects Comput.* **1** (1989) 273–292.

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