Model Checking with Finite Complete Prefixes is PSPACE-complete[⋆]

Keijo Heljanko

Helsinki University of Technology
Laboratory for Theoretical Computer Science
P.O. Box 5400, FIN-02015 HUT, Finland
Keijo.Heljanko@hut.fi

Abstract. Unfoldings are a technique for verification of concurrent and distributed systems introduced by McMillan. The method constructs a finite complete prefix, which can be seen as a symbolic representation of an interleaved reachability graph. We show that model checking a fixed size formula of several temporal logics, including LTL, CTL, and CTL*, is PSPACE-complete in the size of a finite complete prefix of a 1-safe Petri net. This proof employs a class of 1-safe Petri nets for which it is easy to generate a finite complete prefix in polynomial time.

1 Introduction

Unfoldings are a technique for verification of concurrent and distributed systems introduced by McMillan [19]. It can be applied to systems modeled by Petri nets, communicating automata, or process algebras [10, 9, 18]. The method is based on the notion of *unfolding*, which can be seen as the partial order version of an (infinite) computation tree [10, 3].

The unfolding based verification methods construct a *finite complete prefix*, which is a finite initial part of the unfolding containing all the information about the interleaved reachability graph of the system in question. Thus a finite complete prefix can be seen as a symbolic representation of the reachability graph. The finite complete prefix is not, however, a canonical representation of the reachability graph in the same way as a ROBDD (reduced ordered binary decision diagram) is when the variable ordering is fixed [11].

McMillan showed that the deadlock checking problem is NP-complete in the size of a finite complete prefix of a 1-safe Petri net. A small variation of that proof can be used to show that reachability is also NP-complete, see e.g. [13]. Because reachability is PSPACE-complete in the size of a 1-safe Petri net, the prefix generation process mapped this PSPACE-complete problem into (a potentially exponentially larger) NP-complete problem.

[⋆] The financial support of Helsinki Graduate School in Computer Science and Engineering, Academy of Finland (Project 47754), Foundation for Financial Aid at Helsinki University of Technology, Emil Aaltonen Foundation, and Nokia Foundation are gratefully acknowledged.

We will show that model checking a fixed CTL formula containing nested temporal modalities is PSPACE-complete in the size of a finite complete prefix of a 1-safe Petri net. Because model checking a fixed CTL formula is also PSPACE-complete in the size of a 1-safe Petri net [6], using a prefix as input to a model checker does not change the complexity of CTL model checking. The fixed CTL formula we use can be expressed in most temporal logics interpreted over interleaved reachability graphs, and we obtain PSPACE-completeness results for several of them.

Our proof employs a class of 1-safe Petri nets for which it is easy to generate a finite complete prefix in deterministic polynomial time. We will show that the prefixes generated by currently employed prefix generation algorithms (see [10]) can sometimes be exponentially larger than what is allowed by the semantic prefix completeness criterion. We do not know whether these prefixes have some properties which would make model checking using them easier than with prefixes fulfilling only the semantic prefix completeness criterion.

The rest of the paper is structured as follows. First in Sect. 2 we define the Petri net notation used in this paper, followed by the definition of finite complete prefixes. We show in Sect. 3 that it is possible to sometimes create exponentially smaller prefixes than the algorithm of [10]. Next in Sect. 4 we present the main result of this work, a proof of model checking PSPACE-completeness for several logics in the size of a finite complete prefix. We give the conclusions in Sect. 5.

2 Petri Net Definitions

Our aim is to define *finite complete prefixes* as a symbolic representation of a reachability graph of a 1-safe Petri net system. Finite complete prefixes are *branching processes* fulfilling some additional constraints. To define branching processes we introduce *occurrence nets*, which are Petri nets of a restricted form. We do the definitions bottom-up, and begin with basic Petri net notation. We follow mainly the notation of [10].

2.1 Petri Nets

A triple $\langle S, T, F \rangle$ 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 also 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 a node x, denoted by $x^\bullet$, is the set $\{y \in S \cup T \mid F(x, y) = 1\}$. Their generalizations on sets of nodes $X \subseteq S \cup T$ are defined as ${}^\bullet X = \bigcup_{x \in X} {}^\bullet x$, and $X^\bullet = \bigcup_{x \in X} x^\bullet$, respectively.

A *marking* of a net $\langle S, T, F \rangle$ is a mapping $S \mapsto \mathbb{N}$. A marking M is identified with the multi-set which contains $M(s)$ copies of s for every $s \in S$. A 4-tuple $\Sigma = \langle S, T, F, M_0 \rangle$ is a *net system* if $\langle S, T, F \rangle$ is a net and M_0 is a marking of $\langle S, T, F \rangle$. A marking M enables a transition $t \in T$ if $\forall s \in S : F(s, t) \leq M(s)$. If t is enabled, it can *occur* leading to a new marking (denoted $M \xrightarrow{t} M'$), where

M' is defined by $\forall s \in S : M'(s) = M(s) - F(s,t) + F(t,s)$. A marking M_n is *reachable* in Σ if there is an *execution*, i.e. a sequence of transitions $t_1, t_2, \ldots, t_n$ and markings $M_1, M_2, \ldots, M_{n-1}$ such that: $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \ldots M_{n-1} \xrightarrow{t_n} M_n$. A reachable marking is 1-safe if $\forall s \in S : M(s) \leq 1$. A net system Σ is 1-safe if all its reachable markings are 1-safe. In this work we will restrict ourselves to net systems which are 1-safe, have a finite number of places and transitions, and also in which each transition has both nonempty pre- and postsets.

2.2 Occurrence Nets

We use $<_F$ ($\leq_F$) to denote the (reflexive) transitive closure of F. Define $\Sigma = \langle S, T, F \rangle$ to 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 $t_1, t_2 \in T$ such that $t_1 \neq t_2$, ${}^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset$, $t_1 \leq_F x_1$, and $t_2 \leq_F x_2$.

An occurrence net is a net $N = \langle B, E, F \rangle$ such that:

- $\forall b \in B : |{}^\bullet b| \leq 1$,
- F is acyclic, i.e. the irreflexive transitive closure of F is a partial order,
- N is finitely preceded, i.e. for any node x of the net, the set of nodes y such that $y \leq_F x$ is finite, and
- $\forall x \in B \cup E : \neg(x \# x)$.

The elements of B and E are called *conditions* and *events*, respectively. The set $Min(N)$ denotes the set of minimal elements of $<_F$. In this work the minimal elements will be conditions, and thus $Min(N)$ can be seen as an initial marking.

A *configuration* C of an occurrence net is a set of events satisfying:

- If $e \in C$ then $\forall e' \in E : e' \leq_F e$ implies $e' \in C$ (C is causally closed), and
- $\forall e, e' \in C : \neg(e \# e')$ (C is conflict-free).

A *local configuration* $[e]$ of an event e is the set of events e', such that $e' \leq_F e$. A *level* of an event e is the length i of the longest sequence $e_1, e_2, \ldots, e_i$ of events, such that $e_i = e$, and $e_1 <_F e_2 <_F \ldots <_F e_i$. Thus $level(e) = 1$ when ${}^\bullet e \subseteq Min(N)$.

2.3 Branching Processes

Branching processes are "unfoldings" of net systems and were introduced by Engelfriet [4]. Let $N_1 = \langle S_1, T_1, F_1 \rangle$ and $N_2 = \langle S_2, T_2, F_2 \rangle$ be two nets. A *homomorphism* h is a mapping $S_1 \cup T_1 \mapsto S_2 \cup T_2$ such that: $h(S_1) \subseteq S_2$, $h(T_1) \subseteq T_2$, and for all $t \in T_1$, the restriction of h to ${}^\bullet t$ is a bijection between ${}^\bullet t$ and ${}^\bullet h(t)$, and similarly for $t^\bullet$ and $h(t)^\bullet$. A *branching process* of a net system $\Sigma = \langle S, T, F, M_0 \rangle$ is a tuple $\beta = \langle N', h \rangle$, where $N' = \langle B', E', F' \rangle$ is an occurrence net, and h is a homomorphism from N' to $\langle S, T, F \rangle$ such that: the restriction of h to $Min(N')$ is a bijection between $Min(N')$ and M_0, and $\forall e_1, e_2 \in E'$, if ${}^\bullet e_1 = {}^\bullet e_2$ and $h(e_1) = h(e_2)$, then $e_1 = e_2$. Thus h maps the conditions and events of an occurrence net to the places and transitions of the corresponding net system in a way which respects the initial marking and the labeling of the transitions and their pre- and postsets.

The marking of Σ associated with a configuration C of β is denoted by $Mark(C) = h((Min(N) \cup C^{\bullet}) \setminus {}^{\bullet}C)$. A configuration of the branching process always corresponds to a reachable marking of Σ. It is shown in [4] that a net system has a maximal branching process up to isomorphism, called the *unfolding*. If the net system has some infinite behavior, the unfolding will also be infinite.

2.4 Finite Complete Prefixes

We now define *finite complete prefixes*:

Definition 1. *A finite branching process β of a net system Σ is* a finite complete prefix *if for each reachable marking M of Σ there exists a configuration C of β, such that:*

- *$Mark(C) = M$, and*
- *for every transition t enabled in M there exists a configuration $C \cup \{e\}$ of β, such that $e \notin C$ and $h(e) = t$.*

A finite complete prefix contains all the information about the interleaved reachability graph of the net system. Algorithms to obtain a finite complete prefix given a (finite state) net system are presented in [10, 9, 19]. The algorithms will mark some events of the prefix β as special *cut-off events*, which we will denote by events marked with crosses in the figures. The intuition behind cut-off events is that they correspond to repetition of behavior found "earlier" in the prefix. Due to space limitations we direct the interested reader to [10, 9, 19].

3 Compact Finite Complete Prefixes

It is well known that finite complete prefixes can sometimes be exponentially more succinct than the reachability graph of the net system [19]. A simple example of such a family of net systems (with the instance $n = 4$ in Fig. 1) has 2^n reachable markings, while the finite complete prefix is polynomial in the size of the net system. In fact, the finite complete prefixes of this family of net systems are isomorphic to the net system itself. The improved prefix generation algo-

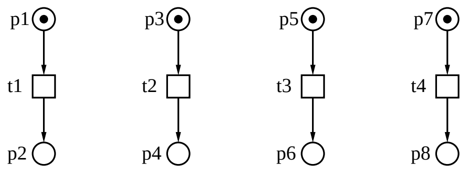

Fig. 1. A 1-safe net system with a compact prefix.

rithm by Esparza, Römer, and Vogler [10] guarantees for 1-safe net systems that the number of non-cut-off events of the generated prefix is never larger than the number of reachable markings. What is not to our knowledge presented in the literature is the fact that sometimes the prefix generation algorithm by McMillan [19] (and also the improved version [10]) creates exponentially larger prefixes than are needed to fulfill the semantic prefix completeness criterion.

For an example of such a family of 1-safe Petri net systems, see Fig. 2. This net system is an instance of a binary counter net system with initialization to a "random" initial state (Fig. 2 is a three bit instance, i.e. $n = 3$). The net system acts like a binary counter starting from all low bits, when the initial marking is $M_0' = \{s(c_0), s(b_0l), s(b_1l), s(b_2l)\}$. The contents of the binary counter are consistent when the place $s(c_0)$ is marked, otherwise the carry propagation can be thought to be in progress. The exact behavior of the net system is actually of no interest to us, we are only interested in the sizes of different finite complete prefixes generated from it.

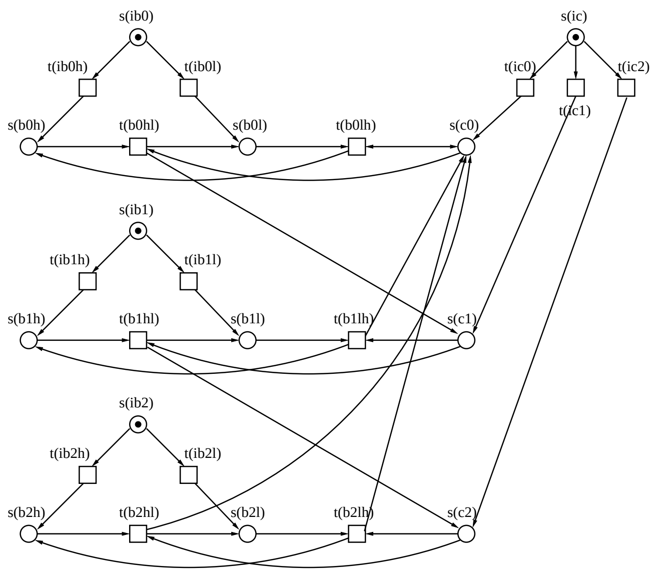

Fig. 2. A three bit binary counter net system.

Let us consider what happens when the initial marking is the marking in Fig. 2. We can find the invariants: $M(s(ic)) + \sum_{0 \leq i < n} M(s(c_i)) = 1$, and for all $0 \leq j < n \, : \, M(s(ib_j)) + M(s(b_jh)) + M(s(b_jl)) = 1$. These $n+1$ invariants give an upper bound of $(n+1) \cdot 3^n$ reachable markings. This is also the exact number of reachable markings, which can also be seen by simple static analysis. Namely, firing one (or none) of the transitions of the set $\{t(ic_0), \ldots, t(ic_{n-1})\}$ can set the first invariant to any of $n+1$ values, and also firing one (or none) of the transitions $\{t(ib_jh), t(ib_jl)\}$ can set the invariant of the bit j into any of three values. Also in the initial state these $n+1$ sets of transitions are concurrently enabled, and thus firing a transition from one set does not disable transitions from other sets. Thus all the $(n+1) \cdot 3^n$ reachable markings are within one concurrent “step” from the initial marking.

We can actually create the finite prefix of Fig. 3 from this net system, and then verify that it fulfills the semantic prefix completeness criterion (Def. 1). We can see that the prefix of Fig. 3 is polynomial in the size of the counter net system of Fig. 2, and that such a “compact prefix” can be constructed for a counter net system of any fixed amount of bits.

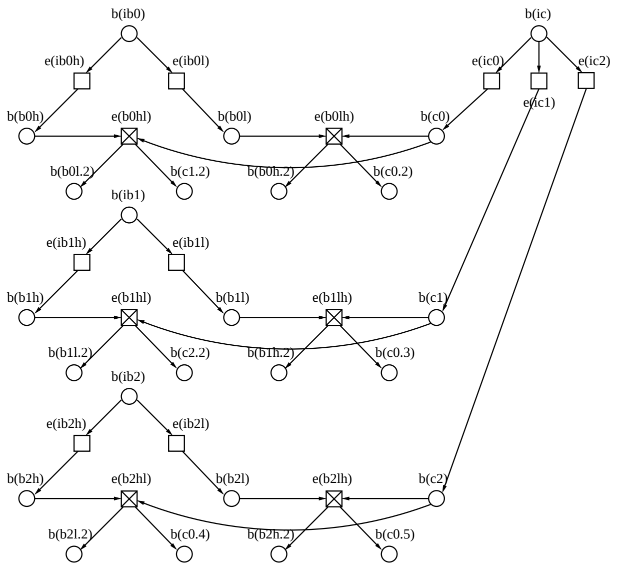

Fig. 3. A finite complete prefix for the three bit counter example.

Here we give a proof sketch for the completeness of the finite prefix in Fig. 3. The prefix is identical to the two first levels of the unfolding of the net system of Fig. 2. The first requirement of prefix completeness is fulfilled, as all of the reachable markings can be reached by a configuration containing only events from the first level of the prefix. The second prefix completeness criterion intuitively requires that all the arcs of the reachability graph are present in the prefix. This is the case, because both the first and second levels of this prefix are identical to the unfolding, and thus they contain extensions for all the configurations (of the first level) mentioned in the second completeness criterion.

Note that all the events on the second level are marked as cut-off events, as they introduce no new reachable markings to the prefix. This requires allowing that the corresponding configuration (see [10]) of a cut-off event is a non-local configuration. Such an idea was already presented in our earlier work [14].

When we consider the sizes of finite complete prefixes generated by the currently employed prefix generation tools, the picture is quite different. We have gathered prefix sizes for small instances of this family of net systems in Table 1. For this family of examples, the McMillan's algorithm [19] and the improvement by Esparza et.al. [10] (implemented in the PEP tool [1]) both generate the same prefixes. While the number of non-cut-off events (the column $|E|-\#\text{c}$ of McMillan/ERV Prefix) grows much more slowly than the number of reachable markings, the growth in this column is still exponential (we get the recursion $x_i = 2x_{i-1}+i+4$, with the initial value $x_2 = 16$). Contrast this with the compact prefix, whose size grows polynomially in the number of bits in the counter. Thus

Table 1. Prefix sizes for counter net systems.

System			Reachability Graph		McMillan/ERV Prefix				Compact Prefix			
Size	$\|S\|$	$\|T\|$	Markings	Arcs	$\|B\|$	$\|E\|$	#c	$\|E\|-\#\text{c}$	$\|B\|$	$\|E\|$	#c	$\|E\|-\#\text{c}$
2	9	10	27	66	43	23	7	16	17	10	4	6
3	13	15	108	351	105	55	16	39	25	15	6	9
4	17	20	405	1620	225	116	30	86	33	20	8	12
5	21	25	1458	6885	453	231	50	181	41	25	10	15
6	25	30	5103	27702	887	449	77	372	49	30	12	18
7	29	35	17496	107163	1721	867	112	755	57	35	14	21

the prefixes generated by the current prefix generation algorithms [19, 10] can be exponentially larger than the compact finite complete prefix which we generated using semantic arguments. This construction relied on the special properties the family of net systems under discussion has. We don't know of a practical algorithm to always generate a polynomial prefix when it is allowed by the semantic notion of prefix completeness.

In the rest of this work we adopt the semantic definition of prefix completeness (Def. 1) as the only property a finite complete prefix has. Thus we can use purely semantic arguments, and do not have to consider the peculiarities of a fixed prefix generation algorithm. However, as presented by Table 1, sometimes the current algorithms will generate exponentially larger prefixes. Thus the complexity results we will present do not automatically apply to these prefixes.

4 Complexity of Model Checking with Complete Prefixes

We can see a 1-safe Petri net system as a representation of its (finite, interleaved) reachability graph. Thus in model checking a Petri net we actually interpret the model checking questions on its reachability graph. Because a finite complete prefix is a symbolic representation of the same reachability graph, we can do model checking when a finite complete prefix is given as input. We will now show that many model checking problems for finite complete prefixes of 1-safe Petri nets are PSPACE-complete in the size of the prefix. This result has been first published in [13], where detailed proofs can be found.

The proof is based on the PSPACE-hardness proof of the reachability problem for 1-safe Petri nets by Jones, Landweber and Lien [15]. The proof involves simulating a Turing machine with a fixed number of tape cells with a 1-safe Petri net. Our proof is based on the variation of this proof by Esparza [6], from which most of the material of the following section is from. We first introduce this proof, because our proof is built on top of it in two steps.

4.1 Reachability with 1-safe Petri nets

We use slightly nonstandard notation in this work. We consider Turing machines with finite tape, i.e. they have both a first and a last cell on their tape. As in the standard definition, a move to the left of the first cell results in the machine staying on the first cell. Slightly nonstandard is the handling of the last cell. If the program of the Turing machine tries to move right when being on the last cell, it stays on the last cell. We define the notions of execution and acceptance of a Turing machine in what is in the essence a standard way, see e.g. [20], with only minor notational differences, for the details see [13].

A *Turing machine* is defined to be a tuple $M = \langle Q, \Gamma, \delta, q_0, F \rangle$, where Q is a finite set of states, Γ a finite set of tape symbols (containing a special *blank* symbol #), $\delta : (Q \times \Gamma) \mapsto \mathcal{P}(Q \times \Gamma \times \{R, L\})$ is the (non-deterministic) transition function, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of final states. We define the *size of a Turing machine* to be the number of bits needed to encode its transition relation, i.e. $2 \cdot |Q|^2 \cdot |\Gamma|^2$.

We define a *linearly bounded automaton* to be a Turing machine which uses n tape cells for a Turing machine description of size n (i.e. the amount of tape matches the size of the transition relation). We encode the configuration of an automaton as $\langle q, i, w \rangle$, where q is the control state of the automaton, $i \in \{1, \ldots, n\}$ is the current location of the tape head, and $w \in \Gamma^n$ is a string of length n which gives the contents of the n tape cells of the automaton. We call a configuration $\langle q, i, w \rangle$ an *initial configuration* if $q = q_0$.

Many question about the computations of linearly bounded automata are PSPACE-hard. The first one we use is the *empty-tape acceptance problem*:

Definition 2. *EMPTY-TAPE ACCEPTANCE:*
Given a linearly bounded automaton $A = \langle Q, \Gamma, \delta, q_0, F \rangle$, does A accept on the empty input?

It is well known that EMPTY-TAPE ACCEPTANCE is PSPACE-complete. Moreover, it remains PSPACE-complete even if we restrict the automaton A to have only one accepting state q_F, see e.g. [6]. We define the size of a 1-safe Petri net system $\Sigma = \langle S, T, F, M_0 \rangle$ to be the number of bits needed to encode the flow relation F, i.e. $\mathcal{O}(|S| \cdot |T|)$. The result we use is the following theorem, first proved by Jones, Landweber and Lien [15]:

Theorem 1. *A linearly bounded automaton of size n can be simulated by a 1-safe Petri net system of size $\mathcal{O}(n^2)$. Moreover, there is a deterministic polynomial time procedure in the size of the automaton which constructs this net.*

We now introduce this mapping from a linearly bounded automaton to a 1-safe Petri net system $N(A)$. See Fig. 4 for an example when $|Q| = 3$, $n = 2$ (smaller than the real n to make the figure smaller), and $\Gamma = \{\#, a, b\}$.

Let $A = \langle Q, \Gamma, \delta, q_0, F \rangle$ be a linearly bounded automaton of size n. We denote the set of tape cells with $C = \{c_1, \ldots, c_n\}$. The simulating Petri net $N(A)$ contains a place $s(q)$ for each state $q \in Q$, a place $s(c_i)$ for each cell $c_i \in C$, and a place $s(a, c_i)$ for each pair $a \in \Gamma, c_i \in C$. A token on place $s(q)$ tells that the machine is in state q, a token on $s(c_i)$ marks the current head location, and when the place $s(a, c_i)$ is marked it means that the symbol on tape cell c_i is a.

The transitions of $N(A)$ are obtained from the transition function of A. If $s(q', a', R) \in \delta(q, a)$, then there exists for each cell $c \in C$ a corresponding transition $t(s(q, a, c)w(a')s(q', c'))$, whose input places are $s(q)$, $s(c)$, and $s(a, c)$, and whose output places are $s(q')$, $s(c')$, and $s(a', c)$, where c' is the cell to the right of c, except when c is the last cell, in which case $c' = c$. The left moves are handled similarly, except that now the first cell is an exception, moving left on it will leave the head on the leftmost cell. The initial marking of $N(A)$ has one token on $s(q_0)$, one on $s(c_1)$, and one on each of the places $s(\#, c_i)$, for all $i \in \{1, \ldots, n\}$. The total size of the net system $N(A)$ is $\mathcal{O}(n^2)$ [6].

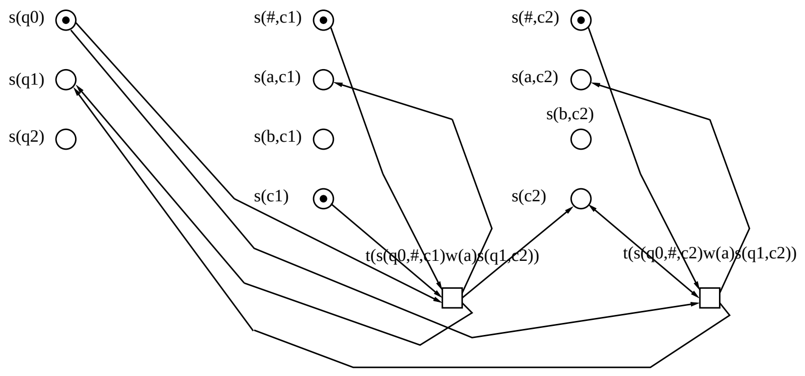

Fig. 4. A part of $N(A)$ simulating a transition $(q_1, a, R) \in \delta(q_0, \#)$.

In this work we use *polynomial-time many-one reductions* (i.e. Karp reductions). Thus given a linearly bounded automaton A with a unique accepting state q_F, we can in deterministic polynomial time construct $N(A)$. Now to decide EMPTY-TAPE ACCEPTANCE we need to answer the following problem on $N(A)$: Is there a reachable marking M of $N(A)$, such that $M(s(q_F)) = 1$?

It is easy to see from the semantics of the branching time temporal logic CTL [3], that the question above is equivalent to the CTL model checking question: $N(A) \models EF(s(q_F))$, i.e. does the formula $EF(s(q_F))$ hold on the interleaved reachability graph of $N(A)$? (We use place names as atomic propositions, e.g. $s(q_F)$ is true in a marking M iff $M(s(q_F)) = 1$.) Thus CTL model checking is

PSPACE-hard in the size of the net system $N(A)$. The model checking problem is also in PSPACE in the size of the 1-safe net system for CTL [6].

4.2 Another PSPACE-hardness Proof with 1-safe Petri Nets

We present an alternative PSPACE-hardness proof for CTL model checking with 1-safe Petri nets. This proof was created to make our proof about prefix model checking complexity (to be presented in the next section) easier.

We use another PSPACE-complete problem for linearly bounded automata:

Definition 3. *ARBITRARY-TAPE-STATE ACCEPTANCE:*
Given a linearly bounded automaton $A = \langle Q, \Gamma, \delta, q_0, F \rangle$ with unique accepting state q_F, does there exists an initial configuration on which A accepts?

In other words: Does there exist an accepting execution of A starting from some initial configuration $\langle q_0, i, w \rangle$, where $i \in \{1, \ldots, n\}$ and $w \in \Gamma^n$?

Theorem 2. *ARBITRARY-TAPE-STATE ACCEPTANCE is PSPACE-complete.*

Proof. See [13]. □

Given a linearly bounded automaton A we will now reduce the problem ARBITRARY-TAPE-STATE ACCEPTANCE into the problem of model checking a certain fixed CTL-formula ϕ on a 1-safe net system $C(A)$. The main intuition behind the reduction is that $C(A)$ is a "cheating simulation" of A, namely it has also behaviors which do not correspond to a simulation of an execution of A. The formula ϕ takes care of ignoring the cheating runs of the net system. We construct a 1-safe Petri net, which first "randomly" initializes the system into some initial state, and then starts to simulate the behavior of the automaton A.

We use the net system $N(A)$ as defined in the previous section as the basis of our mapping, add some places and transitions, and change the initial marking to create a net system $C(A)$ (for details, see [13]). See Fig. 5 for an example of the initialization and simulation of the same transition as in Fig. 4. The places $s(nq), s(nc)$, and $s(nc_i)$ for all $i \in \{1, \ldots, n\}$ are new. They are used to mark that the control state, head location, or contents of the tape cell c_i has not been initialized yet, respectively. For each state $q_i \in Q$ there exists a new transition $t(nq_i)$ whose preset is $s(nq)$ and whose postset is $s(q_i)$. For each tape cell $c_i \in C$ there exists a new transition $t(nc_i)$ whose preset is $s(nc)$ and whose postset is $s(c_i)$. Also for each pair (a, c_i), such that $a \in \Gamma, c_i \in C$, there exists a new transition $t(n(a, c_i))$ whose preset is $s(nc_i)$ and whose postset is $s(a, c_i)$. The initial marking is changed to have a token on the new places added to $C(A)$, and no tokens on other places. This denotes the fact that the initialization needs to be done for state, head location, and each tape cell.

Note that we are even initializing the simulation initial state randomly, instead of initializing it to the state q_0. Thus our simulator is a cheating one. Also

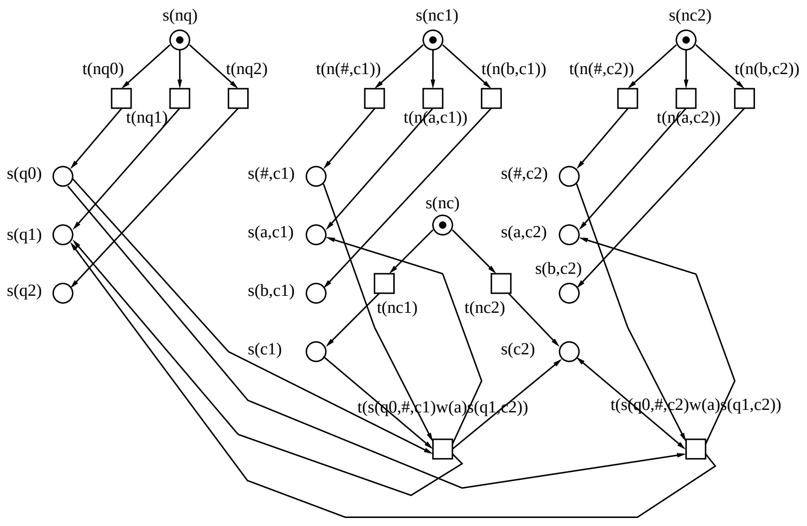

Fig. 5. A cheating Turing machine simulator.

note that the initialization and the beginning of the simulation are not synchronized. This is needed for the prefix to be created to be a compact one, however, it somewhat complicates the proofs in [13].

Lemma 1. *The net system $C(A)$ is 1-safe.*

Proof. The net system $C(A)$ has the following marking invariants:

- $M(s(nq)) + \sum_{q_i \in Q} M(s(q_i)) = 1$,
- $M(s(nc)) + \sum_{c_i \in C} M(s(c_i)) = 1$, and
- for all $i \in \{1, \dots, n\} : M(s(nc_i)) + \sum_{a \in \Gamma} M(s(a, c_i)) = 1$.

The invariants cover all the places of the net system $C(A)$, thus it is 1-safe. □

Now we can show PSPACE-completeness by model checking the CTL formula $\phi = EF(s(q_0) \wedge EF(s(q_F)))$ on the net system $C(A)$.

Lemma 2. *Let A be a linearly bounded automaton with a unique accepting state q_F. It holds that $C(A) \models EF(s(q_0) \wedge EF(s(q_F)))$ iff A has an accepting execution starting from some initial configuration of A.*

Proof. The idea of the proof in one direction is to take an accepting execution of A, and transform it to an execution of $C(A)$, which first fires $n+2$ initialization transitions and then starts simulating the execution of A, giving a witness for the formula ϕ. The other direction is a bit more involved. Whenever $C(A)$ has an execution which is a witness of ϕ, it actually also has an execution which first fires $n + 2$ initialization transitions, and then starts simulating (an accepting execution of) A. Proving this requires a number of lemmas about (a particular kind of) commutativity between the initialization and simulation transitions of $C(A)$. For more details, see [13]. □

Theorem 3. *Model checking a fixed size CTL formula ϕ is PSPACE-complete in the size of the 1-safe net Σ.*

Proof. To show PSPACE-hardness we use the Lemma 2 to reduce the problem ARBITRARY-TAPE-STATE ACCEPTANCE to the problem of CTL model checking a fixed size formula $\phi = EF(s(q_0) \wedge EF(s(q_F)))$ on the net system $C(A)$. The size of $C(A)$ is $\mathcal{O}(n^2)$, i.e. polynomial in the size of A, and the reduction can be done in deterministic polynomial time.

The problem is in PSPACE by Lemma 1, combined with the proof of CTL model checking being in PSPACE in the size of 1-safe net system, see e.g. [6]. □

4.3 Model Checking with Finite Complete Prefixes

We will now make use of the machinery created in the previous sections. We will prove model checking complexity results for algorithms which are given a finite complete prefix of a 1-safe Petri net as the input.

We use the net system $C(A)$ as our starting point, and define the prefix $\beta_C(A)$ to be identical to the first two levels of the unfolding of $C(A)$ (see [13] for the formal definition). For an example of the prefix, see Fig. 6, which is the prefix of the net system in Fig. 5. The prefix $\beta_C(A)$ contains exactly as many

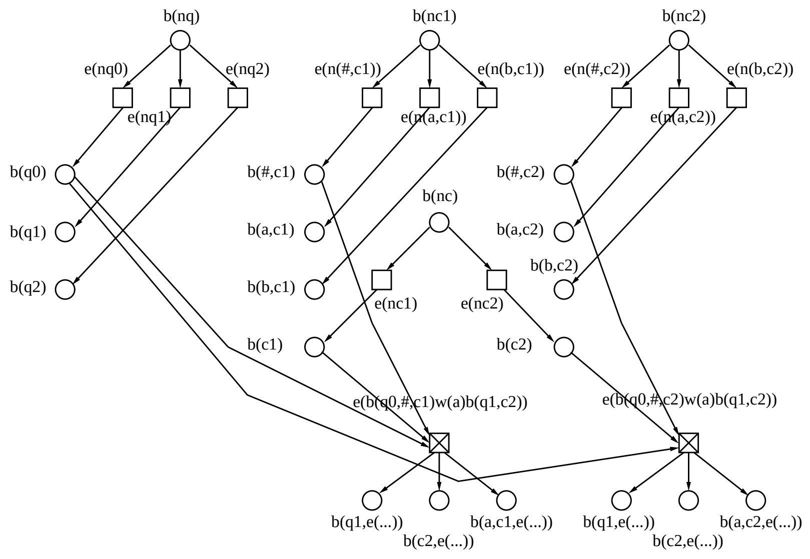

Fig. 6. A finite complete prefix of the cheating Turing machine simulator.

events as there are transitions in $C(A)$. Only the conditions in the postsets of the transitions corresponding to the simulation transitions are new, and there are at most $6 \cdot |Q|^2 \cdot |\Gamma|^2$ of them. Therefore the size of the prefix $\beta_C(A)$ is polynomial in the size of $C(A)$ (and thus also in size of A).

Lemma 3. *The prefix $\beta_C(A)$ is a finite complete prefix of the net system $C(A)$.*

Proof. See [13]. $\square$

Now we can present the main result of this work.

Theorem 4. *Model checking a fixed size CTL formula ϕ is PSPACE-complete in the size of a finite complete prefix β of a 1-safe net Σ.*

Proof. See [13] for details. To show PSPACE-hardness we use the reduction used in the proof of Theorem 3, and then reduce this problem further to CTL model checking ϕ with $\beta_C(A)$ by creating $\beta_C(A)$ in deterministic polynomial time from $C(A)$. Thus by Lemma 3 we get the PSPACE-hardness result.

To show that the problem is in PSPACE in the size of the prefix we use the fact that given a prefix β of a 1-safe net system Σ, and a formula ϕ, we can in polynomial space construct a net system Σ' (by folding the acyclic prefix back into a cyclic net system in the labelling respecting way). For this net system it holds that $\Sigma' \models \phi$ iff $\Sigma \models \phi$. Then we CTL model check $\Sigma' \models \phi$ in PSPACE [6] for a total complexity of PSPACE. $\square$

The CTL formula $\phi = EF(s(q_0) \wedge EF(s(q_F)))$ syntactically belongs to all the logics UB$^-$, UB, CTL, and CTL* (see [3], for the UB logics see e.g. [8]). Therefore the PSPACE-hardness result also applies to them.

We will now require without loss of generality that all executions of the automaton A entering the final state q_F will keep on looping back to the final state q_F thus creating an infinite execution in which the final state is repeated.

We can then create the linear temporal logic LTL (see [3]) formula $\psi = \Box(\neg(s(q_0)) \vee \Box(\neg(s(q_F))))$. Now it is easy to see from the semantics of LTL that $C(A) \models \phi$ iff $C(A) \not\models \psi$. A violation of this LTL formula can be expressed by a Büchi automaton, which can be translated into an equivalent linear-time μ-calculus formula (see e.g. [2]). The LTL formula ψ is also a syntactic safety formula, and thus a violation of this property can also be expressed by a deterministic finite automaton [17]. Thus we get a PSPACE-hardness result for LTL model checking, Büchi emptiness checking, linear-time μ-calculus model checking, and safety model checking.

The model checking problems mentioned above are in PSPACE in the size of the 1-safe net system, and thus we can use the proof of Theorem 4 also with them (see [6], for CTL* we create a concurrent program from a 1-safe Petri net in deterministic polynomial time, and then use a similar result presented for concurrent programs in e.g. [16]). Therefore these model checking problems are PSPACE-complete in the size of a finite complete prefix of a 1-safe Petri net.

5 Conclusions

We have shown that model checking a fixed size formula of several temporal logics, including LTL, CTL, and CTL*, is PSPACE-complete in the size of a finite complete prefix of a 1-safe Petri net. This is to be contrasted with the reachability problem, in which a PSPACE-complete problem for 1-safe Petri nets is transformed by the prefix generation process into (a potentially exponentially larger) NP-complete problem, see e.g. [13]. However, such a drop in complexity (assuming NP is easier than PSPACE) does not occur in the case of model checking involving nested temporal modalities. Thus, loosely speaking, with prefixes reachability is easier than "repeated reachability" (see [13]).

Our proof employs a class of 1-safe Petri nets for which it is easy to create a finite complete prefix by using semantic arguments. We have shown that sometimes the prefixes created by current prefix generation algorithms [10] will be exponentially larger than allowed by the semantic completeness criterion (Def. 1). The definition of a suitable prefix minimality criterion, and the creation of a procedure to always obtain these compact prefixes is left for further work.

There are prefix based model checkers which handle nested temporal modalities. The LTL model checker of [21] creates a certain graph, whose size can be exponential in the size of the prefix. The construction employed by the branching time model checker of [5, 12] to handle nested temporal modalities is more involved, and relating our work to the results of [12] is left for further study. We would also like to know whether the prefixes generated by [10] have some properties which would allow simpler model checking algorithms than the prefixes fulfilling only the semantic prefix completeness criterion. Finally, for LTL model checking we can change the model checker to take both the net system, and the LTL(-X) formula ψ as input to the prefix generation process. In this approach also the semantic prefix completeness criterion is parameterized by the checked formula, and the model checking can be done in polynomial time in the size of this "product" prefix [7]. The price to pay is a larger prefix. A simpler product method works with safety model checking.

Acknowledgements

Part of this work was done during a visit to Prof. J. Esparza's research group at TU München. The author would like to thank for the visit opportunity, and for discussions about the unfolding method. The author would also like to thank Ilkka Niemelä and Tommi Junttila for critical comments on this work.

References

[1] E. Best. Partial order verification with PEP. In *Proceedings of POMIV'96, Workshop on Partial Order Methods in Verification*. American Mathematical Society, July 1996.

[2] M. Dam. Fixpoints of Büchi automata. In *Proceedings of the 12th International Conference of Foundations of Software Technology and Theoretical Computer Science*, pages 39–50, 1992. LNCS 652.

[3] E. A. Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science, Volume B, Formal Models and Semantics*, pages 995–1072. North-Holland Pub. Co./MIT Press, 1990.

[4] J. Engelfriet. Branching processes of Petri nets. *Acta Informatica*, 28:575–591, 1991.

[5] J. Esparza. Model checking using net unfoldings. *Science of Computer Programming*, 23(2):151–195, 1994.

[6] J. Esparza. Decidability and complexity of Petri net problems – An introduction. In *Lectures on Petri Nets I: Basic Models*, pages 374–428. Springer-Verlag, 1998. LNCS 1491.

[7] J. Esparza and K. Heljanko. A new unfolding approach to LTL model checking. In *Proceedings of 27th International Colloquium on Automata, Languages and Programming (ICALP'2000)*, July 2000. Accepted for publication.

[8] J. Esparza and M. Nielsen. Decidability issues for Petri Nets - a survey. *Journal of Information Processing and Cybernetics 30(3)*, pages 143–160, 1994.

[9] J. Esparza and S. Römer. An unfolding algorithm for synchronous products of transition systems. In *Proceedings of the 10th International Conference on Concurrency Theory (Concur'99)*, pages 2–20. Springer-Verlag, 1999. LNCS 1664.

[10] J. Esparza, S. Römer, and W. Vogler. An improvement of McMillan's unfolding algorithm. In *Proceedings of Second International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96)*, pages 87–106. Springer-Verlag, March 1996. LNCS 1055.

[11] J. Feigenbaum, S. Kannan, M. Y. Vardi, and M. Viswanathan. Complexity of problems on graphs represented as OBDDs. *Chigago Journal of Theoretical Computer Science*, 1999(5):1–25, 1999.

[12] B. Graves. Computing reachability properties hidden in finite net unfoldings. In *Proceedings of 17th Foundations of Software Technology and Theoretical Computer Science Conference*, pages 327–341. Springer-Verlag, 1997. LNCS 1346.

[13] K. Heljanko. Deadlock and reachability checking with finite complete prefixes. Research Report A56, Helsinki University of Technology, Laboratory for Theoretical Computer Science, Espoo, Finland, December 1999. Licentiate's Thesis. Available at http://www.tcs.hut.fi/pub/reports/A56.ps.gz.

[14] K. Heljanko. Minimizing finite complete prefixes. In *Proceedings of the Workshop Concurrency, Specification & Programming 1999*, pages 83–95. Warsaw University, Warsaw, Poland, September 1999.

[15] N. D. Jones, L. H. Landweber, and Y. E. Lien. Complexity of some problems in Petri nets. *Theoretical Computer Science*, 4:277–299, 1977.

[16] O. Kupferman. *Model Checking for Branching-Time Temporal Logics*. PhD thesis, Technion, Israel Institute of Technology, Haifa, Israel, June 1995.

[17] O. Kupferman and M. Y. Vardi. Model checking of safety properties. In *Proceeding of 11th International Conference on Computer Aided Verification (CAV'99)*, pages 172–183. Springer-Verlag, 1999. LNCS 1633.

[18] R. Langerak and E. Brinksma. A complete finite prefix for process algebra. In *Proceeding of 11th International Conference on Computer Aided Verification (CAV'99)*, pages 184–195. Spriger-Verlag, 1999. LNCS 1633.

[19] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[20] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[21] F. Wallner. Model checking techniques using net unfoldings. PhD thesis, Technische Universität München, Germany, forthcoming.