A New Unfolding Approach to LTL Model Checking *

Javier Esparza[1] and Keijo Heljanko[2]

[1] Institut für Informatik, Technische Universität München, Germany
e-mail: esparza@in.tum.de
[2] Lab. for Theoretical Computer Science, Helsinki University of Technology, Finland
e-mail: Keijo.Heljanko@hut.fi

Abstract A new unfolding approach to LTL model checking is presented, in which the model checking problem can be solved by direct inspection of a certain finite prefix. The techniques presented so far required to run an elaborate algorithm on the prefix.

1 Introduction

Unfoldings are a partial order technique for the verification of concurrent and distributed systems, initially introduced by McMillan [11]. They can be understood as the extension to communicating automata of the well-known unfolding of a finite automaton into a (possibly infinite) tree. The unfolding technique can be applied to systems modelled by Petri nets, communicating automata, or process algebras [4,3,10]. It has been used to verify properties of circuits, telecommunication systems, distributed algorithms, and manufacturing systems [1].

Unfoldings have proved to be very suitable for deadlock detection and invariant checking [11]. For these problems, one first constructs a so-called *complete prefix* [4], a finite initial part of the unfolding containing all the reachable states. This prefix is at most as large as the state space, and usually much smaller (often exponentially smaller). Once the prefix has been constructed, the deadlock detection problem can be easily reduced to a graph problem [11], an integer linear programming problem [12], or to a logic programming problem [8].

In [2,7] and [17,16], unfolding-based model checking algorithms have been proposed for a simple branching-time logic and for LTL, respectively. Although the algorithms have been applied with success to a variety of examples, they are not completely satisfactory: After constructing the complete prefix, the model checking problem cannot be yet reduced to a simple problem like, say, finding cycles in a graph. In the case of LTL the intuitive reason is that the infinite sequences of the system are "hidden" in the finite prefix in a complicated way. In order to make them "visible", a certain graph has to be constructed. Unfortunately, the graph can be exponentially larger than the complete prefix itself.

* Work partially supported by the Teilprojekt A3 SAM of the Sonderforschungsbereich 342 "Werkzeuge und Methoden für die Nutzung paralleler Rechnerarchitekturen", the Academy of Finland (Project 47754), and the Nokia Foundation.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 475–486, 2000.

Niebert has observed [13] that this exponential blow-up already appears in a system of n independent processes, each of them consisting of an endless loop with one single action as body. The complete prefix has size $\mathcal{O}(n)$, which in principle should lead to large savings in time and space with respect to an interleaving approach, but the graph is of size $\mathcal{O}(2^n)$, i.e. as large as the state space itself.

In this paper we present a different unfolding technique which overcomes this problem. Instead of unrolling the system until a complete prefix has been generated, we "keep on unrolling" for a while, and stop when certain conditions are met. There are two advantages: (i) the model checking problem can be solved by a direct inspection of the prefix, and so we avoid the construction of the possibly exponential graph; and, (ii) the algorithm for the construction of the new prefix is similar to the old algorithm for the complete prefix; only the definition of a cut-off event needs to be changed. The only disadvantage is the larger size of the new prefix. Fortunately, we are able to provide a bound: the prefix of a system with K reachable states contains at most $\mathcal{O}(K^2)$ events, assuming that the system is presented as a 1-safe Petri net or as a product of automata[1]. Notice that this is an upper bound: the new prefix is usually much smaller than the state space, and in particular for Niebert's example it grows linearly in n.

The paper is structured as follows (for detailed definitions and proofs see the full version [5]). Section 2 presents the automata theoretic approach to LTL model checking. In Sect. 3 the unfolding method is introduced. Sections 4 and 5 contain the tableau systems for the two subproblems. In Sect. 6 we show how LTL model checking can be solved with the presented tableau systems. In Sect. 7 we conclude and discuss topics for further research.

2 Automata theoretic approach to model checking LTL

Petri nets. We assume that the reader is familiar with basic notions, such as net, preset, postset, marking, firing, firing sequence, and reachability graph. We consider labelled nets, in which places and transitions carry labels taken from a finite alphabet $\mathcal{L}$, and labelled net systems. We denote a labelled net system by $\Sigma = (P, T, F, l, M_0)$, where P and T are the sets of places and transitions, F is the flow function $F\colon (P \times T) \cup (T \times P) \to \{0, 1\}$, $l\colon P \cup T \to \mathcal{L}$ is the labelling function, and M_0 is the initial marking.

We present how to modify the automata theoretic approach to model checking LTL [15] to best suit the net unfolding approach. For technical convenience we use an action-based temporal logic instead of a state-based one, namely the linear temporal logic $tLTL'$ of Kaivola, which is immune to the stuttering of invisible actions [9]. With small modifications the approach can also handle state based stuttering invariant logics such as LTL-X. Given a finite set A of actions, and a set $V \subseteq A$ of visible actions, the abstract syntax of $tLTL'$ is given by:

$$\varphi ::= \top \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \,\mathcal{U}\, \varphi_2 \mid \varphi_1 \,\mathcal{U}^a\, \varphi_2, \text{ where } a \in V$$

[1] More precisely, the number of non-cut-off events is at most $\mathcal{O}(K^2)$.

Formulas are interpreted over sequences of A^ω. The semantics of $\varphi_1 \mathcal{U} \varphi_2$ is as expected. Loosely speaking, a sequence w satisfies $\varphi_1 \mathcal{U}^a \varphi_2$ if φ_1 holds until the first a in w, and then φ_2 holds[2].

Given a net system $\Sigma = (P, T, F, l, M_0)$, where the transitions of T are labelled with actions from the set A, and a formula φ of $tLTL'$, the model checking problem consists of deciding if all the infinite firing sequences of Σ satisfy φ.

The *automata theoretic approach* attacks this problem as follows. First, a procedure similar to that of [6] converts the *negation* of φ into a Büchi automaton $\mathcal{A}_{\neg\varphi}$ over the alphabet $\Gamma = V \cup \{\tau\}$, where $\tau \notin A$ is a new label used to represent all the invisible actions. Then, this automaton is synchronized with Σ *on visible actions* (see [5] for details). The synchronization can be represented by a new labelled net system $\Sigma_{\neg\varphi}$ containing a transition (u, t) for every $u = q \xrightarrow{a} q'$ in $\mathcal{A}_{\neg\varphi}$ and for every $t \in T$, such that $l(t) = a$ and $a \in V$, plus other transitions for the invisible transitions of Σ. We say that (u, t) is an *infinite-trace monitor* if q' is a final state of $\mathcal{A}_{\neg\varphi}$, and a *livelock monitor* if the automaton $\mathcal{A}_{\neg\varphi}$ accepts an infinite sequence of invisible transitions (a *livelock*) with q' as initial state. The sets of infinite-trace and livelock monitors are denoted by I and L, respectively. An *illegal ω-trace* of $\Sigma_{\neg\varphi}$ is an infinite firing sequence $M_0 \xrightarrow{t_1 t_2 \ldots}$ such that $t_i \in I$ for infinitely many indices i. An *illegal livelock* of $\Sigma_{\neg\varphi}$ is an infinite firing sequence $M_0 \xrightarrow{t_1 t_2 \ldots t_i} M \xrightarrow{t_{i+1} t_{i+2} \ldots}$ such that $t_i \in L$, and $t_{i+k} \in (T \setminus V)$ for all $k \geq 1$. We have the following result:

Theorem 1. *Let Σ be a labelled net system, and φ a $tLTL'$-formula. $\Sigma \models \varphi$ if and only if $\Sigma_{\neg\varphi}$ has no illegal ω-traces and no illegal livelocks.*

The intuition behind this theorem is as follows. Assume that Σ can execute an infinite firing sequence corresponding to a word $w \in (V \cup \{\tau\})^\omega$ violating φ (where 'corresponding' means that the firing sequence executes the same visible actions in the same order, and an invisible action for each τ). If w contains infinitely many occurrences of visible actions, then $\Sigma_{\neg\varphi}$ contains an illegal ω-trace; if not, it contains an illegal livelock.

In the next sections we provide unfolding-based solutions to the problems of detecting illegal ω-traces and illegal livelocks. We solve the problems in an abstract setting. We fix a net system $\Sigma = (P, T, F, M_0)$, where T is divided into two sets V and $T \setminus V$ of *visible* and *invisible* transitions, respectively. Moreover, T contains two special subsets L and I. We assume that no reachable marking of Σ concurrently enables a transition of V and a transition of L. We further assume that M_0 does not put more than one token on any place. In particular, when applying the results to the model checking problem for $tLTL'$ and Petri nets, the system Σ is the synchronization $\Sigma_{\neg\varphi}$ of a Petri net and a Büchi automaton, and it satisfies these conditions. We use as running example the net system of Fig. 1. We have $V = \{t_6\}$, $I = \{t_1\}$, and $L = \{t_2\}$. The system has illegal ω-traces (for instance, $(t_1 t_3 t_4 t_6 t_7)^\omega$), but no illegal livelocks.

[2] Kaivola's semantics is interpreted over $A^* \cup A^\omega$, which is a small technical difference.

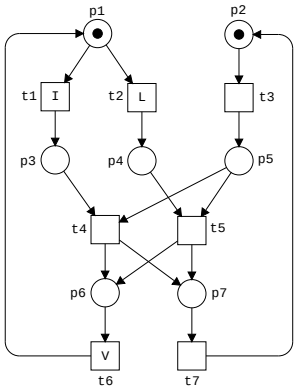

Figure 1. A net system

3 Basic definitions on unfoldings

In this section we briefly introduce the definitions we need to describe the unfolding approach to our two problems. More details can be found in [4].

Occurrence nets. Given two nodes x and y of a net, we say that x is *causally related* to y, denoted by $x \leq y$, if there is a path of arrows from x to y. We say that x and y are in *conflict*, denoted by $x\#y$, if there is a place z, different from x and y, from which one can reach x and y, exiting z by different arrows. Finally, we say that x and y are *concurrent*, denoted by $x \, co \, y$, if neither $x \leq y$ nor $y \leq x$ nor $x\#y$ hold. A *co-set* is a set of nodes X such that $x \, co \, y$ for every $x, y \in X$. *Occurrence nets* are those satisfying the following three properties: the net, seen as a graph, has no cycles; every place has at most one input transition; and, no node is in self-conflict, i.e., $x\#x$ holds for no x. A place of an occurrence net is *minimal* if it has no input transitions. The net of Fig. 2 is an infinite occurrence net with minimal places a, b. The *default initial marking* of an occurrence net puts one token on each minimal place an none in the rest.

Branching processes. We associate to Σ a set of *labelled* occurrence nets, called the *branching processes* of Σ. To avoid confusions, we call the places and transitions of branching processes *conditions* and *events*, respectively. The conditions and events of branching processes are labelled with places and transitions of Σ, respectively. The conditions and events of the branching processes are subsets from two sets $\mathcal{B}$ and $\mathcal{E}$, inductively defined as the smallest sets satisfying:

- $\perp \in \mathcal{E}$, where $\perp$ is an special symbol;
- if $e \in \mathcal{E}$, then $(p, e) \in \mathcal{B}$ for every $p \in P$;
- if $\emptyset \subset X \subseteq \mathcal{B}$, then $(t, X) \in \mathcal{E}$ for every $t \in T$.

In our definitions we make consistent use of these names: The label of a condition (p, e) is p, and its unique input event is e. Conditions $(p, \perp)$ have no

input event, i.e., the special symbol $\perp$ is used for the minimal places of the occurrence net. Similarly, the label of an event (t, X) is t, and its set of input conditions is X. The advantage of this scheme is that a branching process is completely determined by its sets of conditions and events. We make use of this and represent a branching process as a pair (B, E).

Definition 1. *The set of* finite branching processes *of a net system Σ with the initial marking $M_0 = \{p_1, \ldots, p_n\}$ is inductively defined as follows:*

- $(\{(p_1, \perp), \ldots, (p_n, \perp)\}, \emptyset)$ *is a branching process of* Σ.[3]
- *If (B, E) is a branching process of Σ, $t \in T$, and $X \subseteq B$ is a co-set labelled by ${}^\bullet t$, then $(\, B \cup \{(p, e) \mid p \in t^\bullet\} \,,\, E \cup \{e\}\,)$ is also a branching process of Σ, where $e = (t, X)$. If $e \notin E$, then e is called a* possible extension *of (B, E).*

The set of branching processes of Σ is obtained by declaring that the union of any finite or infinite set of branching processes is also a branching process, where union of branching processes is defined componentwise on conditions and events. Since branching processes are closed under union, there is a unique maximal branching process, called the *unfolding* of Σ. The unfolding of our running example is an infinite occurrence net. Figure 2 shows an initial part. Events and conditions have been assigned identificators that will be used in the examples. For instance, the event $(t_1, \{(p_1, \perp)\})$ is assigned the identificator 1.

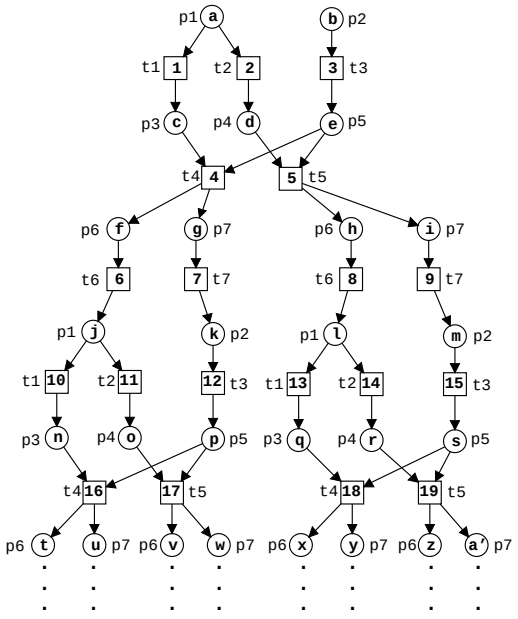

Figure 2. The unfolding of Σ

[3] This is the point at which we use the fact that the initial marking is 1-safe.

Configurations. A *configuration* of an occurrence net is a set of events C satisfying the two following properties: C is causally closed, i.e., if $e \in C$ and $e' < e$ then $e' \in C$, and C is conflict-free, i.e., no two events of C are in conflict. Given an event e, we call $[e] = \{e' \in E \mid e' \leq e\}$ the *local configuration* of e. Let *Min* denote the set of minimal places of the branching process. A configuration C of the branching process is associated with a marking of Σ denoted by $Mark(C) = l((Min \cup C^\bullet) \setminus {}^\bullet C)$.

In Fig. 2, $\{1, 3, 4, 6\}$ is a configuration, and $\{1, 4\}$ (not causally closed) or $\{1, 2\}$ (not conflict-free) are not. A set of events is a configuration if and only if there is one or more firing sequences of the occurrence net (from the default initial marking) containing each event from the set exactly once, and no further events. These firing sequences are called *linearisations*. The configuration $\{1, 3, 4, 6\}$ has two linearisations, namely 1 3 4 6 and 3 1 4 6. All linearisations lead to the same reachable marking. For example, the two sequences above lead to the marking $\{p_1, p_7\}$. By applying the labelling function to a linearisation we obtain a firing sequence of Σ. Abusing of language, we also call this firing sequence a linearisation. In our example we obtain $t_1t_3t_4t_6$ and $t_3t_1t_4t_6$ as linearisations.

Given a configuration C, we denote by $\uparrow C$ the set of events $e \in E$, such that: (1) $e' < e$ for some event $e' \in C$, and (2) e is not in conflict with any event of C. Intuitively, $\uparrow C$ corresponds to the behavior of Σ from the marking reached after executing any of the linearisations of C. We call $\uparrow C$ the *continuation* after C of the unfolding of Σ. If C_1 and C_2 are two finite configurations leading to the same marking, i.e. $Mark(C_1) = M = Mark(C_2)$, then $\uparrow C_1$ and $\uparrow C_2$ are isomorphic, i.e., there is a bijection between them which preserves the labelling of events and the causal, conflict, and concurrency relations (see [4]).

4 A tableau system for the illegal ω-trace problem

In this section we present an unfolding technique for detecting illegal ω-traces. We introduce it using the terminology of tableau systems, the reason being that the technique has many similarities with tableau systems as used for instance in [18] for model-checking LTL, or in [14] for model-checking the mu-calculus. However, no previous knowledge of tableau systems is required.

Adequate orders. We need the notion of *adequate order* on configurations [4]. In fact, our tableau system will be parametric in the adequate order, i.e., we will obtain a different system for each adequate order. Given a configuration C of the unfolding of Σ, we denote by $C \oplus E$ the set $C \cup E$, under the condition that $C \cup E$ is a configuration satisfying $C \cap E = \emptyset$. We say that $C \oplus E$ is an *extension* of C. Now, let C_1 and C_2 be two finite configurations leading to the same marking. Then $\uparrow C_1$ and $\uparrow C_2$ are isomorphic. This isomorphism, say f, induces a mapping from the extensions of C_1 onto the extensions of C_2; the image of $C_1 \oplus E$ under this mapping is $C_2 \oplus f(E)$.

Definition 2. *A partial order $\prec$ on the finite configurations of the unfolding of a net system is an* adequate order *if:*

- $\prec$ *is well-founded,*
- $C_1 \subset C_2$ *implies* $C_1 \prec C_2$*, and*
- $\prec$ *is preserved by finite extensions; if* $C_1 \prec C_2$ *and* $Mark(C_1) = Mark(C_2)$*, then the isomorphism* f *from above satisfies* $C_1 \oplus E \prec C_2 \oplus f(E)$ *for all finite extensions* $C_1 \oplus E$ *of* C_1*.*

Total adequate orders are particularly good for our tableau systems because they lead to stronger conditions for an event to be a terminal, and so to smaller tableaux. Total adequate orders for 1-safe Petri nets and for synchronous products of transition systems, have been presented in [4,3].

4.1 The tableau system

Given a configuration C of the unfolding of Σ, denote by $\#_I C$ the number of events $e \in C$ labelled by transitions of I.

Definition 3. *An event* e *of a branching process* BP *is a* repeat *(with respect to* $\prec$*) if* BP *contains another event* e'*, called the* companion *of* e*, such that* $Mark([e']) = Mark([e])$*, and either*

(I) $e' < e$*, or*
(II) $\neg(e' < e)$*,* $[e'] \prec [e]$*, and* $\#_I[e'] \geq \#_I[e]$*.*

A terminal *is a minimal repeat with respect to the causal relation; in other words, a repeat* e *is a terminal if the unfolding of* Σ *contains no repeat* $e' < e$*. Repeats, and in particular terminals, are of type I or type II, according to the condition they satisfy.*

Events labelled by I*-transitions are called* I*-events. A repeat* e *with companion* e' *is* successful *if it is of type I, and* $[e] \setminus [e']$ *contains some* I*-event. Otherwise it is* unsuccessful*.*

A tableau *is a branching process* BP *such that for every possible extension* e *of* BP *at least one of the immediate causal predecessors of* e *is a terminal. A tableau is* successful *if at least one of its terminals is successful.*

Loosely speaking, a tableau is a branching process which cannot be extended without adding a causal successor to a terminal. In the case of a terminal of type I, $\uparrow[e]$ need not be constructed because $\uparrow[e']$, which is isomorphic to it, will be in the tableau. In the case of a terminal of type II, $\uparrow[e]$ need not be constructed either, because $\uparrow[e']$ will appear in the tableau. However, in order to guarantee completeness, we need the condition $\#_I[e'] \geq \#_I[e]$.

The tableau construction is straightforward. Given $\Sigma = (N, M_0)$, where $M_0 = \{p_1, \ldots, p_n\}$, start from the branching process $(\{(p_1, \perp), \ldots, (p_n, \perp)\}, \emptyset)$. Add events according to the inductive definition of branching process, but with the restriction that no event having a terminal as a causal predecessor is added. Events are added in $\prec$ order; more precisely, if $[e] \prec [e']$, then e is added before e'. The construction terminates when no further events can be added.

We construct the tableau corresponding to the net system of Fig. 1 using the total adequate order of [4].[4] All we need to know about this order is that for the events 4 and 5 in Fig. 2, $[4] \prec [5]$ holds. The tableau is the fragment of the unfolding of Fig. 2 having events 16, 17, and 5 as terminals. Events 16 and 17 are terminals of type I having event 4 as companion. Event 16 is successful because the set $[16] \setminus [4] = \{6, 7, 10, 11, 12, 16\}$ contains an I-event, namely 10. The intuition behind these terminals is rather clear: a terminal of type I corresponds to a cycle in the reachability graph. Loosely speaking, the events of $[16] \setminus [4]$ correspond to a firing sequence leading from $Mark([4])$ to $Mark([16])$, and these two markings coincide. Since $[16] \setminus [4]$ contains an I-event, the firing sequence contains a transition of I, and so we have found an illegal ω-trace. The set $[17]\setminus[4]$ doesn't contains any I-event, but $\uparrow[17]$ need not be constructed, because it is isomorphic to $\uparrow[4]$. Event 5 is a terminal of type II with event 4 as companion because $Mark([4]) = \{p_6, p_7\} = Mark([5])$, $[4] \prec [5]$, and $1 = \#_I[4] \geq \#_I[5] = 0$. The intuition is that $\uparrow[5]$ need not be constructed, because it is isomorphic to $\uparrow[4]$. However, this doesn't explain why the condition $\#_I[e'] \geq \#_I[e]$ is needed. In [5] we present an example showing that after removing this condition the tableau system is no longer complete.

Let K denote the number of reachable markings of Σ, and let B denote the maximum number of tokens that the reachable markings of Σ put in all the places of Σ. We have the following result:

Theorem 2. *Let $\mathcal{T}$ be a tableau of Σ constructed according to a total adequate order $\prec$.*

- *$\mathcal{T}$ is successful if and only if Σ has an illegal ω-trace.*
- *$\mathcal{T}$ contains at most $K^2 \cdot B$ non-terminal events.*
- *If the transitions of I are pairwise non-concurrent, then $\mathcal{T}$ contains at most K^2 non-terminal events.*

5 A tableau system for the illegal livelock problem

The tableau system for the illegal livelock problem is a bit more involved that that of the illegal ω-trace problem. In a first step we compute a set $CP = \{M_1, \ldots, M_n\}$ of reachable markings of Σ, called the set of *checkpoints*. This set has the following property: if Σ has an illegal livelock, then it also has an illegal livelock $M_0 \xrightarrow{t_1 t_2 \ldots t_i} M \xrightarrow{t_{i+1} t_{i+2} \ldots}$ such that $t_i \in L$ and M is a checkpoint. For the computation of CP we use the unfolding technique of [4] or [3]; the procedure is described in Sect. 5.1.

The tableau system solves the problem whether some checkpoint enables an infinite sequence of invisible actions. Clearly, Σ has an illegal livelock if and only if this is indeed the case. For this, we consider the net N_{inv} obtained from N by removing all the visible transitions together with their adjacent arcs. We construct unfoldings for the net systems $(N_{inv}, M_1), \ldots, (N_{inv}, M_n)$, and check

[4] We can also take the order of [3], which for this example yields the same results.

on them if the systems exhibit some infinite behavior. The tableau system is described in Sect. 5.2.

5.1 Computing the set of checkpoints.

We construct the complete prefix of the unfolding of Σ as defined in [4] or [3]. In the terminology of this paper, the complete prefix corresponds to a tableau in which an event e is a terminal if there is an event e' such that $Mark([e']) = Mark([e])$, and $[e'] \prec [e]$.

Definition 4. *A marking M belongs to the set CP of* checkpoints *of Σ if $M = Mark([e])$ for some non-terminal event e of the complete prefix of Σ labelled by a transition of L.*

Let us compute CP for our example. The complete prefix of Σ coincides with the tableau for the illegal ω-trace problem. The events labelled by t_2, the only transition of L, are 2 and 11. The corresponding markings are $Mark([2]) = \{p_2, p_4\}$ and $Mark([11]) = \{p_4, p_7\}$. So $CP = \{\, \{p_2, p_4\}, \{p_4, p_7\} \,\}$.

5.2 The tableau system

Let $\{M_1, \ldots, M_n\}$ be the set of checkpoints obtained in the first phase. We will use $\Sigma_1, \ldots, \Sigma_n$ to denote the net systems $(N_{inv}, M_1), \ldots, (N_{inv}, M_n)$.

Definition 5. *Let $BP_1, \ldots, BP_n$ be branching processes of $\Sigma_1, \ldots, \Sigma_n$, respectively. An event e of BP_i is a* repeat *(with respect to $\prec$) if there is an index $j \leq i$ and an event e' in BP_j, called the* companion *of e, such that $Mark([e']) = Mark([e])$, and either*

(I) $j < i$, *or*
(II) $i = j$ *and* $e' < e$, *or*
(III) $i = j$, $\neg(e' < e)$, $[e'] \prec [e]$, *and* $|[e']| \geq |[e]|$.

A repeat e of BP_i is a terminal *if BP_i contains no repeat $e' < e$. Repeats, and in particular terminals, are of type I, II, or III, according to the condition they satisfy. A repeat e with companion e' is* successful *if it is of type II, and* unsuccessful *otherwise.*

A tableau *is a tuple $BP_1, \ldots, BP_n$ of branching processes of $\Sigma_1, \ldots, \Sigma_n$ such that for every $1 \leq i \leq n$ and for every possible extension e of BP_i at least one of the immediate causal predecessors of e is a terminal. Each BP_i is called a* tableau component. *A tableau is* successful *if at least one of its terminals is successful.*

Observe that an event of BP_i can be a repeat because of an event that belongs to *another* branching process BP_j. The definition of repeat depends on the order of the checkpoints, but the tableau system defined above is sound and complete for any fixed order. Because the definition of the tableau component

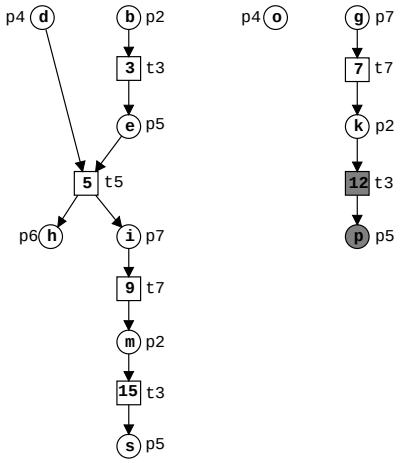

Figure 3. The tableau system for the illegal livelock problem

BP_i depends only on the components with a smaller index, we can create the tableau components in increasing i order. Tableau components are constructed as for the illegal ω-trace problem, using the new definition of terminal.

The tableau for our example is shown in Fig. 3. The names of places and transitions have been chosen to match "pieces" of the unfolding in Fig. 2. The first tableau component contains no terminals; the construction terminates because no event labelled by an invisible transition can be added. In the second component, event 12 is a terminal with event 3 in the first component as companion. The intuition is that we don't need to unfold beyond 12 in the second component, because what we construct can be found after 3 in the first component.

Similarly to the case of the illegal ω-trace problem, a terminal of type II corresponds to a cycle in the reachability graph. Since the transitions of N_{inv} are all invisible, such a cycle always originates an illegal livelock, and so terminals of type II are always successful. For terminals of type III, the intuition is that $\uparrow[e]$ need not be constructed, because it is isomorphic to $\uparrow[e']$. The condition $|[e']| \geq |[e]|$ is required for completeness (see [5]). We have the following result:

Theorem 3. *Let $\mathcal{T}_1, \ldots, \mathcal{T}_n$ be a tableau of $\Sigma_1, \ldots, \Sigma_n$ constructed according to a total adequate order $\prec$.*

- *$\mathcal{T}_1, \ldots, \mathcal{T}_n$ is successful if and only if Σ contains an illegal livelock.*
- *$\mathcal{T}_1, \ldots, \mathcal{T}_n$ contain together at most $K^2 \cdot B$ non-terminal events.*

5.3 A tableau system for the 1-safe case

If Σ is 1-safe then we can modify the tableau system to obtain a bound of K^2 non-terminal events. We modify the definition of the repeats of type II and III:

(II') $i = j$ and $\neg(e' \# e)$, or

(III') $i = j$, $e' \# e$, $[e'] \prec [e]$, and $|[e']| \geq |[e]|$.

Theorem 4. *Let Σ be 1-safe. Let $\mathcal{T}_1, \ldots, \mathcal{T}_n$ be a tableau of $\Sigma_1, \ldots, \Sigma_n$ constructed according to a total adequate order $\prec$, and to the new definition of repeats of type II and III.*

- *$\mathcal{T}_1, \ldots, \mathcal{T}_n$ is successful if and only if Σ contains an illegal livelock.*
- *$\mathcal{T}_1, \ldots, \mathcal{T}_n$ contain together at most K^2 non-terminal events.*

6 A tableau system for LTL model checking

Putting the tableau systems of Sections 4 and 5 together, we obtain a tableau system for the model checking problem of $tLTL'$. For the sake of clarity we have considered the illegal ω-trace problem and the illegal livelock problem separately. However, when implementing the tableau systems there is no reason to do so. Since all the branching processes we need to construct are "embedded" in the unfolding of $\Sigma_{\neg\varphi}$, it suffices in fact to construct *one single branching process*, namely the union of all the processes needed to solve both problems.

Clearly, this prefix contains $\mathcal{O}(K^2 \cdot B)$ non-terminal events. If the system is presented as a 1-safe Petri net, then the prefix contains $\mathcal{O}(K^2)$ non-terminal events because the following two conditions hold: (i) None of the reachable markings of the synchronization $\Sigma_{\neg\phi}$ enable two I-transitions concurrently. (ii) If the system is a 1-safe Petri net, then the synchronization $\Sigma_{\neg\phi}$ is also 1-safe.

7 Conclusions

We have presented a new unfolding technique for checking LTL-properties. We first make use of the automata-theoretic approach to model checking: a combined system is constructed as the product of the system itself and of an automaton for the negation of the property to be checked. The model checking problem reduces to the illegal ω-trace problem and to the illegal livelock problem for the combined system. Both problems are solved by constructing certain prefixes of the net unfolding of the combined system. In fact, it suffices to construct the union of these prefixes.

The prefixes can be seen as tableau systems for the illegal ω-trace and the illegal livelock problem. We have proved soundness and completeness of these tableau systems, and we have given an upper bound on the size of the tableau. For systems presented as 1-safe Petri nets or products of automata, tableaux contain at most size $\mathcal{O}(K^2)$ (non-terminal) events, where K is the number of reachable states of the system. An interesting open problem is the existence of a better tableau system such that tableaux contain at most $\mathcal{O}(K)$ events. We conjecture that it doesn't exist.

The main advantage of our approach is its simplicity. Wallner's approach proceeds in two steps: construction of a complete prefix, and then construction of a graph. The definition of a graph is non-trivial, and the graph itself can be exponential in the size of the complete prefix. Our approach makes the construction of the graph unnecessary. The price to pay is a larger prefix.

References

1. Bibliography on the net unfolding method. Available on the Internet at `http://wwwbrauer.in.tum.de/gruppen/theorie/pom/pom.shtml`.
2. J. Esparza. Model checking using net unfoldings. *Science of Computer Programming*, 23:151–195, 1994.
3. 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, 1999. LNCS 1055.
4. J. Esparza, S. Römer, and W. Vogler. An improvement of McMillan's unfolding algorithm. In *Proceedings of 2nd International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96)*, pages 87–106, 1996. LNCS 1055.
5. Javier Esparza and Keijo Heljanko. A new unfolding approach to LTL model checking. Research Report A60, Helsinki University of Technology, Laboratory for Theoretical Computer Science, Espoo, Finland, April 2000. Available at `http://www.tcs.hut.fi/pub/reports/A60.ps.gz`.
6. R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Proceedings of 15th Workshop Protocol Specification, Testing, and Verification*, pages 3–18, 1995.
7. 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, 1997. LNCS 1346.
8. K. Heljanko. Using logic programs with stable model semantics to solve deadlock and reachability problems for 1-safe Petri nets. In *Proceedings of 5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99)*, pages 240–254, 1999. LNCS 1579.
9. R. Kaivola. Using compositional preorders in the verification of sliding window protocol. In *Proceeding of 9th International Conference on Computer Aided Verification (CAV'97)*, pages 48–59, 1997. LNCS 1254.
10. 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, 1999. LNCS 1663.
11. K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.
12. S. Melzer and S. Römer. Deadlock checking using net unfoldings. In *Proceedings of 9th International Conference on Computer-Aided Verification (CAV '97)*, pages 352–363, 1997. LNCS 1254.
13. P. Niebert. Personal communication, 1999.
14. C. Stirling and David Walker. Local model checking in the modal mu-calculus. *Theoretical Computer Science*, 89(1):161–177, 1991.
15. M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Logics for Concurrency: Structure versus Automata*, pages 238–265, 1996. LNCS 1043.
16. F. Wallner. Model checking techniques using net unfoldings. PhD thesis, Technische Universität München, Germany, forthcoming.
17. F. Wallner. Model checking LTL using net unfoldings. In *Proceeding of 10th International Conference on Computer Aided Verification (CAV'98)*, pages 207–218, 1998. LNCS 1427.
18. P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56(1,2):72–93, 1983.