Petri Net Analysis and Nonmonotonic Reasoning

Keijo Heljanko* and Ilkka Niemelä*

Abstract

The paper presents a symbolic analysis method for solving bounded deadlock detection and reachability questions for Petri nets using nonmonotonic reasoning techniques. More precisely, a mapping is devised such that given a 1-safe P/T-net, some Boolean conditions on the initial marking, and a bound n, a logic program is obtained such that there is an execution of at most n steps of the net starting from some initial marking satisfying the conditions leading to deadlock iff the logic program has a stable model. A similar mapping is given for reachability questions from a set of initial markings satisfying given Boolean conditions. Experiments to solve deadlock problems using the `Smodels` system as the stable model finder indicate that the approach can provide a competitive method for finding short executions to deadlocks.

1 Introduction

This is a paper for which the research started 15 years ago. It combines two areas that seem to have very little to do with each other. The idea is to employ nonmonotonic reasoning techniques for analyzing Petri nets. We start the paper by an account of its history. This provides a nice example of the way Prof. Leo Ojala does research and leads a research group. It also illustrates the open-mindedness, far-sightedness and long-term commitment of Leo in his approach to research.

The second author began to work on this paper (without even realizing it) in spring 1985 in a postgraduate seminar organized by Leo. The topic was nonmonotonic logics. They involved strange mathematic concepts like fixed point equations and second-order logic. It was hard to understand the definitions and even harder to comprehend where all this could be used. Gradually it became clearer (to the second author) that these logics are being developed to solve some fundamental problems in artificial intelligence and, in particular, in knowledge representation. The problems arise when describing dynamic systems, e.g., effects of actions, using a logic-based

*Both from Helsinki University of Technology, Dept. of Computer Science and Engineering, Laboratory for Theoretical Computer Science, P.O.Box 5400, FIN-02015 HUT, Finland. The financial support of Helsinki Graduate School in Computer Science and Engineering, Academy of Finland (Projects 43963 and 47754), Foundation for Financial Aid at Helsinki University of Technology, Emil Aaltonen Foundation, and Nokia Foundation are gratefully acknowledged.

approach. The claim was that classical logics are not well suited for this because of, e.g., the *frame problem* [15] and new nonmonotonic logics are needed.

Leo and his student Heikki Tuominen had got interested in nonmonotonic reasoning in their study of using logic in Petri net analysis. One approach was to describe the dynamics of a Petri net axiomatically employing a suitable logic. They had used temporal logic [1] but this was not quite straightforward, e.g., because of the frame problem and they had started with nonmonotonic logics as a possible remedy. However, very little was known about the complexity, decidability and applicability of such odd logics. As many young people do, the second author got interested in such a challenging topic and started his Master's thesis on the subject in 1986.

Sorting out the basic properties of this new form of reasoning kept the second author busy and we did not get very far in axiomatizing Petri nets using nonmonotonic logics. However, other interesting things got done. For the axiomatic approach Heikki decided to use dynamic logic and gave an axiomatization of elementary net systems [28]. Also a model checking approach was developed [28, 25] where the reachability graph of a Petri net was interpreted as a (Kripke) model of temporal logic which was used as a query language for checking interesting properties of the net. Gradually such a model checker has become a standard part of the Petri net analysis tools developed in Leo's laboratory [29, 8, 14].

It became clear quite early that nonmonotonic logics were not ripe for applications and a lot of work was needed. This did not discourage Leo and he let the second author to continue the research on nonmonotonic reasoning. This was a new area and novel results were obtained starting from decidability [18], computational complexity [19], and decision procedures [21]. The innocent little side-track of Petri net research grew gradually and has resulted already in, e.g., four doctoral dissertations [20, 26, 11, 27].

In the early nineties the results obtained indicated that it could be possible to automate nonmonotonic logics (suitably restricted) as efficiently as classical logics and maybe efficiently enough to make them interesting for real applications. A serious effort to demonstrate the applicability of nonmonotonic reasoning techniques started and led to the development of the `Smodels` system [23, 24, 27] (`http://www.tcs.hut.fi/Software/smodels/`) whose first version was released in summer 1995. It turned out that the nonmonotonic techniques in `Smodels` are promising in many areas such as constraint satisfaction, combinatorial problems and planning [22].

The maturing implementation technology for nonmonotonic reasoning provided an opportunity to apply it seriously to Petri net analysis. The first approach was to use net unfoldings (finite complete prefixes) [10, 9]. `Smodels` was used here as a solver for reachability and deadlock questions on prefixes with encouraging results. Hence, it felt very interesting to return to the problem where all work on nonmonotonic reasoning started in Leo's lab 15 years earlier, i.e., axiomatizing Petri nets using nonmonotonic logics. The idea is to describe the behavior of a Petri net as well as the property to be analyzed symbolicly using the nonmonotonic formalism supported by `Smodels` in such a way that `Smodels` could analyze the property using directly the symbolic representation of the net and, e.g., without constructing its reachability graph.

In the meanwhile significant progress has been made in using the symbolic approach in system verification. The first breakthrough was in hardware verification where symbolic model checking techniques proved to be very efficient [3, 4]. This approach is based on encoding reachable states using Boolean formulae represented by (ordered) binary decision diagrams (BDDs). Although the BDD-based approach has been successfully, there are difficulties in applying the technique, in particular, in areas outside hardware verification. The key problem is that some Boolean functions do not have a compact representation as BDDs and the size of the BDD representation of a Boolean function is very sensitive to the variable ordering used for constructing the BDD.

Bounded model checking [2] has been proposed as a technique for overcoming the space problem by replacing BDDs with propositional satisfiability (SAT) checking techniques because typical SAT checkers use only polynomial amount of memory. The idea is roughly the following. Given a sequential digital circuit, a (temporal) property to be verified, and a bound n, the behavior of the circuit is unfolded up to n steps as a Boolean formula S and the negation of the property to be verified is represented as a Boolean formula R. The translation to Boolean formulae is done so that $S \wedge R$ is satisfiable iff the system has a behavior violating the property of length at most n. A main advantage of the bounded model checking approach is that it can find fast counterexamples, i.e., behaviors violating the correctness requirements. When searching for the counterexamples by increasing gradually the bound n, one finds those of minimal length. This helps the user to understand the counterexamples more easily.

Until now bounded model checking has been applied to synchronous hardware verification. In this work we extend the approach to handle asynchronous concurrent systems modeled by Petri nets. It turns out that bounded model checking for 1-safe P/T-nets is closely related to AI planning. Here we show how to map bounded model checking problems to the problem of finding stable models of logic programs by employing ideas used in reducing planning to stable model computation [22].

The structure of the rest of the paper is the following. In the next section we introduce Petri nets and the bounded model checking problem. Then we present the nonmonotonic logic supported by `Smodels` (logic programs with the stable model semantics) which we employ in the following section to achieve a compact encoding of bounded model checking. We present some experimental results and finish with some concluding remarks.

2 Petri nets and bounded model checking

We will now introduce P/T-nets. They are one of the simplest forms of Petri nets. We will use as a running example the P/T-net represented in Figure 1.

A triple $\langle P, T, F \rangle$ is a *net* if $P \cap T = \emptyset$ and $F \subseteq (P \times T) \cup (T \times P)$. The elements of P are called *places*, and the elements of T *transitions*. Places and transitions are also called *nodes*.

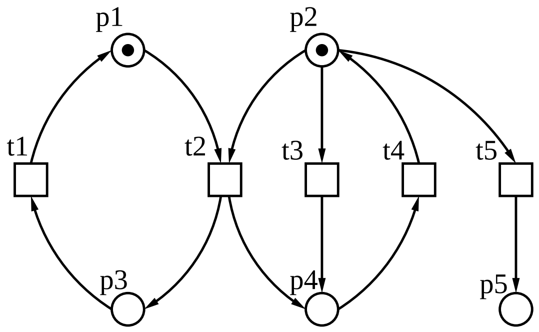

Figure 1: Running Example: A 1-safe P/T-net

We identify F with its characteristic function on the set $(P \times T) \cup (T \times P)$. The *preset* of a node x, denoted by ${}^\bullet x$, is the set $\{y \in P \cup T \mid F(y, x) = 1\}$. The *postset* of a node x, denoted by $x^\bullet$, is the set $\{y \in P \cup T \mid F(x, y) = 1\}$.

A *marking* of a net $\langle P, T, F\rangle$ is a mapping $P \mapsto \mathbb{N}$. A marking M is identified with the multi-set which contains $M(p)$ copies of p for every $p \in P$. A 4-tuple $\Sigma = \langle P, T, F, M_0\rangle$ is a *net system* (also called a *P/T-net*) if $\langle P, T, F\rangle$ is a net and M_0 is a marking of $\langle P, T, F\rangle$.

A marking M enables a transition $t \in T$ if $\forall p \in P : F(p, t) \leq M(p)$. If t is enabled, it can *occur* leading to a new marking (denoted $M \xrightarrow{t} M'$), where M' is defined by $\forall p \in P : M'(p) = M(p) - F(p, t) + F(t, p)$. In the running example the transition $t2$ is enabled in the initial marking M_0, and thus $M_0 \xrightarrow{t2} M'$, where $M' = \{p3, p4\}$.

A marking M_n is *reachable* in Σ if there is an *execution*, i.e. a (possibly empty) 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 marking M is reachable within a bound n, if there is an execution with at most n transitions, with which M is reachable from the initial state.

A marking is 1-safe if $\forall p \in P : M(p) \leq 1$. A P/T-net Σ is 1-safe if all its reachable markings are 1-safe. In this work we will restrict ourselves to P/T-nets which are 1-safe, have a finite number of places and transitions, and in which each transition has both nonempty pre- and postsets.

Given a 1-safe P/T-net Σ, we say that a set of transitions $S \subseteq T$ is *concurrently enabled* in the marking M, if (i) all transitions $t \in S$ are enabled in M, and (ii) for all pairs of transitions $t, t' \in S$, such that $t \neq t'$, it holds that ${}^\bullet t \cap {}^\bullet t' = \emptyset$. If a set S is concurrently enabled in the marking M, we can fire it in a *step* (denoted $M \xrightarrow{S} M'$), where M' is the marking reached after firing all of the transitions in the step S in arbitrary order. In our running example in the marking $M' = \{p3, p4\}$ the step $\{t1, t4\}$ is enabled, and will lead back to the initial marking M_0. This is denoted by $M' \xrightarrow{\{t1,t4\}} M_0$.

We say that a marking M_n is *reachable in step semantics* in a 1-safe P/T-net Σ if there is an *step execution*, i.e. a (possibly empty) sequence of steps $S_1, S_2, \ldots, S_n$ and markings $M_1, M_2, \ldots, M_{n-1}$ such that: $M_0 \xrightarrow{S_1} M_1 \xrightarrow{S_2} \ldots M_{n-1} \xrightarrow{S_n} M_n$. A marking M is reachable within a bound n in the step semantics if there is a step execution with at most n steps, with which M is reachable from the initial state. We will often refer to the "normal semantics" as *interleaving semantics* to more clearly distinguish it from the step semantics. We have the following theorem.

Theorem 1 *For 1-safe P/T-nets the set of reachable markings in the interleaving semantics and the set of reachable markings in the step semantics coincide.*

Thus the step semantics does not bring any new reachable markings. However, it allows to fire several transitions "in one time step". Such *concurrency* implies that the number of "time steps" that the system is executed can be decreased without losing any reachable markings.

Reachability and deadlock detection are among the most important problems in the analysis of Petri net models.

Definition 1 (Reachability) *Given a 1-safe P/T-net Σ and a 1-safe marking M, is M a reachable marking of Σ?*

Definition 2 (Deadlock) *Given a 1-safe P/T-net Σ, is there a reachable marking M, which does not enable any transition of Σ?*

Both the reachability and deadlock problems for 1-safe Petri nets are PSPACE-complete [12, 6].

In the bounded case there are now two problems and two different semantics to consider. We will define only one of them, the others are defined in a similar fashion.

Definition 3 (Bounded deadlock, step semantics) *Given a 1-safe P/T-net Σ and an integer bound $n \geq 0$, is there a marking M reachable within the bound n in the step semantics such that M does not enable any transition of Σ?*

We can think about the bounded versions of the problems as approximations of the original problems, which become increasingly better as the bound n increases. The main motivation is that if we find a solution to the bounded version, then the original problem also has that solution.

We will now define the notion of a *reachability diameter* for both semantics, which is the semantic version of the "sufficient bound":

Definition 4 (Reachability diameter) *Given a 1-safe P/T-net Σ, the reachability diameter d for the step (interleaving) semantics is the smallest integer $d \geq 0$ such that the set of reachable markings and the set of reachable markings in the step (interleaving) semantics within bound d coincide.*

See [2] for discussion on how to obtain a reachability diameter using a QBF formula (using a slightly different definition of the diameter, however, the discussion still applies here). In practice the currently used tools do not support the calculation of the diameter for examples of interesting size. Therefore, bounded model checking is at its best in "bug hunting", and not as easily applicable in verifying systems to be correct.

3 Nonmonotonic reasoning

Nonmonotonic logics formalize a pattern of reasoning which is not supported by any classical logic: adding new premises might invalidate previous conclusions. All classical logics are monotonic meaning that new premises can only increase the set of conclusions. A general approach to formalizing nonmonotonic reasoning is to see it as *autoepistemic reasoning* [17], i.e., as reasoning performed by a fully introspective agent. The set of premises is given using a modal language containing an operator L which refers to the beliefs of the agent. Besides classical consequences the agent is capable of performing *positive introspection* (if it believes ϕ, then it believes $L\phi$) and *negative introspection* (if it does not believe ϕ, then it believes $\neg L\phi$). It is negative introspection that makes the reasoning nonmonotonic.

Given a set of premises Σ, a possible set of beliefs of a fully introspective agent, called a *stable expansion* Δ of Σ, is defined by the following a fixed point equation:

$$\Delta = \{\psi \mid \Sigma \cup L\Delta \cup \neg L\overline{\Delta} \models \psi\} \tag{1}$$

which says that Δ is a possible set of beliefs if it is the set of formulae which are classical consequences ($\models$) of the premises Σ, positive introspection $L\Delta = \{L\phi \mid \phi \in \Delta\}$ and negative introspection $\neg L\overline{\Delta} = \{\neg L\phi \mid \phi \notin \Delta\}$. A possible set of nonmonotonic conclusions from a set of premises can be taken as a set of beliefs (a stable expansion) of a fully introspective agent given the premises.

Autoepistemic logic provides a unified basis for solving many kinds of knowledge representation issues [20]. For example, consider the frame problem, i.e., the problem of compactly representing conditions saying that things remain the same unless something forces them to change. It can be handled using frame axioms of the form

$$p(T) \wedge \neg Lt_1(T) \wedge \cdots \wedge \neg Lt_l(T) \rightarrow p(T+1) \tag{2}$$

saying that if proposition p holds in a situation T and none of the transitions $t_1, \ldots, t_l$ capable of changing p are not believed to be executed, then p holds also in the next situation $T+1$.

General autoepistemic logic is very expressive and consequently computationally hard [19]. For most engineering problems, it is enough to restrict to a subset capable of capturing **NP**-complete problems. For this it is sufficient to consider autoepistemic formulae of the form

$$a_1 \wedge \cdots \wedge a_m \wedge \neg La_{m+1} \wedge \cdots \wedge \neg La_n \rightarrow a_0 \tag{3}$$

where every a_i is an atomic formula. Such a formula states a simple constraint on a stable expansion saying that is each of $a_1, \ldots, a_m$ is included in the expansion but none of $a_{m+1}, \ldots, a_n$ is, then a_0 must be included. Gelfond and Lifschitz [7] pointed out that this fragment of autoepistemic logic provides a very intuitive declarative semantics for normal logic programs where the idea is to map the negation (as failure) in logic programming 'not a' to a disbelief '$\neg La$' in autoepistemic logic. Hence, an autoepistemic formula (3) corresponds to the normal logic program rule

$$a_0 \leftarrow a_1, \ldots, a_m, \text{not } a_{m+1}, \ldots, \text{not } a_n\ . \tag{4}$$

The declarative semantics is obtained by taking as the possible models of the program the atomic parts of the stable expansions of the program seen as the corresponding set of autoepistemic formulae. Such a model, i.e., the set of the atomic formulae in a stable expansion, is called a *stable model* of the program.

It was this fragment of autoepistemic logic corresponding to normal programs that we decided originally to implement in the `Smodels` system. In order to make `Smodels` more expressive and easier to apply, the language of normal programs has been extended with cardinality and weight constraints [24, 27]. For the purpose of this paper it is enough to understand two cases of the extensions. The first is a compact way of encoding a conditional choice using a rule on the left

$$\{a_0\} \leftarrow a_1, \ldots, a_m \qquad\qquad \begin{array}{l} a_0 \leftarrow \text{not } \hat{a_0}, a_1, \ldots, a_m \\ \hat{a_0} \leftarrow \text{not } a_0 \end{array}$$

which says that a_0 can be included in a stable model only if $a_1, \ldots, a_m$ are also included but a_0 can be left out. This corresponds to the two normal rules on the right where a new atom $\hat{a_0}$ has to be introduced as the complement of a_0. The other case

$$\leftarrow 2\{a_1, \ldots, a_m\}$$

excludes all stable models where at least two of $a_1, \ldots, a_m$ are included. There seems to be no compact encoding of such a rule using only a linear number of normal rules.

4 Bounded symbolic model checking

In this section we develop a method for translating bounded model checking problems of 1-safe P/T-nets to tasks of finding stable models. We do this so that model checking can be done for sets of initial markings satisfying certain properties at once.

Consider a P/T-net $N = \langle P, T, F \rangle$ and a step bound n. We construct a logic program $\Pi_{\mathrm{A}}(N, n)$ whose stable models correspond to the possible executions of the net up to n steps in the following way.

- For each place $p \in P$, include a choice rule: $\{p(0)\} \leftarrow$
- For each transition $t \in T$, and for all $i = 0, 1, \ldots, n-1$, include a rule

  $$\{t(i)\} \leftarrow p_1(i), \ldots, p_l(i) \tag{5}$$

  where $\{p_1, \ldots, p_l\}$ is the preset of t. Hence, a stable model can contain a transition instance in step i only if its preset holds at step i.
- For each place $p \in P$ and for all $i = 0, 1, \ldots, n-1$, include a rule

  $$p(i+1) \leftarrow t(i) \tag{6}$$

  for each t in the preset of p. The rules imply that p holds in the next step if at least one of its preset transitions is in the current step.

$$
\begin{array}{ll}
\{p1(0)\} \leftarrow & p3(i+1) \leftarrow t2(i) \\
\{p2(0)\} \leftarrow & p4(i+1) \leftarrow t2(i) \\
\{p3(0)\} \leftarrow & p4(i+1) \leftarrow t3(i) \\
\{p4(0)\} \leftarrow & p5(i+1) \leftarrow t5(i) \\
\{p5(0)\} \leftarrow & \leftarrow 2\{t2(i), t3(i), t5(i)\} \\
\{t1(i)\} \leftarrow p3(i) & p1(i+1) \leftarrow p1(i), \mathrm{not}\ t2(i) \\
\{t2(i)\} \leftarrow p1(i), p2(i) & p2(i+1) \leftarrow p2(i), \mathrm{not}\ t2(i), \mathrm{not}\ t3(i), \mathrm{not}\ t5(i) \\
\{t3(i)\} \leftarrow p2(i) & p3(i+1) \leftarrow p3(i), \mathrm{not}\ t1(i) \\
\{t4(i)\} \leftarrow p4(i) & p4(i+1) \leftarrow p4(i), \mathrm{not}\ t4(i) \\
\{t5(i)\} \leftarrow p2(i) & p5(i+1) \leftarrow p5(i) \\
p1(i+1) \leftarrow t1(i) & \\
p2(i+1) \leftarrow t4(i) & \text{where } i = 0, 1, \ldots n-1
\end{array}
$$

Figure 2: Program $\Pi_{\mathrm{A}}(N, n)$

- For each place $p \in P$, and for all $i = 0, 1, \ldots, n-1$, include a rule

  $$\leftarrow 2\{t_1(i), \ldots, t_l(i)\} \tag{7}$$

  where $\{t_1, \ldots, t_l\}$ is the set of transitions having each p in their preset and $l \geq 2$. This rule states that at most one of the transitions that are in conflict w.r.t. p can occur.

- For each place p, and for all $i = 0, 1, \ldots, n-1$,

  $$p(i+1) \leftarrow p(i), \mathrm{not}\ t_1(i), \ldots, \mathrm{not}\ t_l(i) \tag{8}$$

  where $\{t_1, \ldots, t_l\}$ is the set of transitions having p in their preset. This is the *frame axiom* for p stating that p holds if no transition using it occurs.

Consider our running example. The program $\Pi_{\mathrm{A}}(N, n)$ is given in Figure 2. Conditions on markings are straightforward to state using rules. Eliminating stable models not satisfying a marking M at step i can be achieved using rules

$$
\begin{array}{rcl}
\Pi_{\mathrm{M}}(M, i) & = & \{\leftarrow \mathrm{not}\ p(i) \mid p \in P, M(p) = 1\} \cup \\
 & & \{\leftarrow p(i) \mid p \in P, M(p) = 0\}.
\end{array}
$$

This extends to any Boolean combination directly. For example, for eliminating stable models not satisfying condition C at step i requiring that $M(p_1) = 1$ with $M(p_2) = 0$ or $M(p_3) = 1$ it is sufficient to use rules $\Pi_{\mathrm{M}}(C, i)$:

$$
\begin{array}{ll}
\leftarrow \mathrm{not}\ p_1(i) & c_{\bar{p_2} \vee p_3}(i) \leftarrow \mathrm{not}\ p_2(i) \\
\leftarrow \mathrm{not}\ c_{\bar{p_2} \vee p_3}(i) & c_{\bar{p_2} \vee p_3}(i) \leftarrow p_3(i)
\end{array}
$$

Theorem 2 *Let $N = \langle P, T, F \rangle$ be a 1-safe P/T-net for all initial markings satisfying condition C. Net N has an initial marking satisfying C such that a marking M is reachable in at most n steps iff $\Pi_{\mathrm{M}}(C, 0) \cup \Pi_{\mathrm{A}}(N, n) \cup \Pi_{\mathrm{M}}(M, n)$ has a stable model.*

This approach can be adapted easily to handle deadlock checking by adding rules $\Pi_{\mathrm{D}}(N, n)$ eliminating stable models where some transition is enabled. Program $\Pi_{\mathrm{D}}(N, n)$ includes for each transition $t \in T$, a rule

$$\leftarrow p_1(n), \ldots, p_l(n) \tag{9}$$

where $\{p_1, \ldots, p_l\}$ is the preset of t. For our running example, rules $\Pi_{\mathrm{D}}(N, n)$ are

$$\begin{array}{ll} \leftarrow p3(n) & \leftarrow p2(n) \\ \leftarrow p1(n), p2(n) & \leftarrow p4(n). \end{array}$$

Theorem 3 *Let $N = \langle P, T, F \rangle$ be a 1-safe P/T-net for all initial markings satisfying condition C. Net N has an initial marking satisfying C such that a deadlock is reachable in at most n steps iff $\Pi_{\mathrm{M}}(C, 0) \cup \Pi_{\mathrm{A}}(N, n) \cup \Pi_{\mathrm{D}}(N, n)$ has a stable model.*

So far in this section we have considered only the translations of the step semantics versions of the problems. We can create the interleaving semantics versions of the problems by adding some rules to the program for the step version of the problem. Program $\Pi_{\mathrm{I}}(N, n)$ includes for each time step $0 \leq i \leq n-1$ a rule

$$\leftarrow 2\{t_1(i), \ldots, t_m(i)\} \tag{10}$$

where $\{t_1, \ldots, t_m\}$ is the set of all transitions. These rules will the eliminate all stable models having more than one transition firing in a step.

Corollary 1 *Let $\Pi_{\mathrm{S}}(N, n)$ be a translation solving a bounded model checking problem in the step semantics. Then the program $\Pi_{\mathrm{S}}(N, n) \cup \Pi_{\mathrm{I}}(N, n)$ solves the same problem in the interleaving semantics.*

In [2] it is shown how bounded model checking can be done also for linear time temporal logic LTL. An interesting area of further work is to extend bounded model checking of LTL formulae to the asynchronous case. A main challenges is to allow as much concurrency as possible, to obtain as small as possible diameter for the LTL model checking translation. Also the safety property subset of LTL is interesting in this context [13], as a simpler translation for that LTL subset is possible.

5 Experiments

We have implemented the translation from the bounded model checking problem to the problem of finding a stable model. The translation was implemented in C++ in quite a straightforward manner with only two simple optimizations included:

- If a single initial marking is given, place and transition atoms are added only from the time step they can first appear on. Only atoms for places $p(0)$ in the initial marking are created for time $i = 0$. Then for each $0 \leq i \leq n-1$: (i) Add transition atoms for all transitions $t(i)$ such that all the place atoms in the preset of $t(i)$ exist. (ii) Add place atoms for all places $p(i+1)$ such that either the place atom $p(i)$ exists or some transition atom in the preset of $p(i+1)$ exists.

- Duplicate rules are removed (can appear in conflict (7) and liveness (9) rules).

As benchmarks we use a set of deadlock checking problems collected by Corbett [5]. They have been converted from communicating state machines to 1-safe P/T-nets by Melzer and Römer [16]. The problem is to check deadlocks from a given single initial marking. The testcases were picked from those which have a deadlock, and `Smodels` was instructed to stop after finding the first stable model using the smallest bound n in which the deadlock existed. In some cases a model could not be found within a reasonable time, in which case we report the time used to prove that there is no deadlock within the bound n.

The experimental results can be found in Fig. 3 with the following columns:

- Problem: The problem name, with the size of the instance in parenthesis.
- $|P|$ ($|T|$): Number of places (transitions) in the original net.
- St. n: The smallest integer n such that a deadlock could be found using the step semantics / in case of $> n$ the largest integer n for which we could prove that there is no deadlock within that bound using the step semantics.
- St. s: The time in seconds to find the first stable model / to prove that there is no stable model. (See St. n above.)
- Int. n and Int. s: defined as St. n and St. s but for the interleaving semantics.
- States: Number of reachable states of the P/T-net (if known).

The times are the average of 5 runs of the time for `Smodels 2.26` as reported by the `/usr/bin/time` command on a 450Mhz Pentium III PC running Linux.

In many of the experiments the step semantics version had a much smaller bound than the interleaving one. Also, when the bound needed to find the deadlock was fairly small, the bounded model checker was performing well.

The DP(x) problems are dining philosophers problems, where in the step semantics the counterexample could always be found with a bound of 1, while in the interleaving semantics the bound grew at the same speed as the number of philosophers. In the examples ELEV(4), HART(x) and Q(1) we were able to find the counterexample only when using step semantics.

In the KEY(2) example we were no able to find a counterexample with either semantics, even though the problem is known to have only a small number of reachable states. In contrast, the DARTES(1) problem has a large state-space, and despite of it a counterexample of length 32 was obtained. Thus is seems that the size of the state space is not always decisive in the bounded model checker running time.

This is the first set of experiments we have tried with asynchronous system benchmarks, and no major work has gone into obtaining the best possible performance. Overall, the results were promising, in particular, for small bounds and the step semantics but we need to get a better understanding of the behavior of the bounded model checking approach by doing more experiments.

Problem	$\lvert P\rvert$	$\lvert T\rvert$	St. n	St. s	Int. n	Int. s	States
DARTES(1)	331	257	32	0.5	32	0.5	>250000
DP(6)	36	24	1	0.0	6	0.1	728
DP(8)	48	32	1	0.0	8	0.3	6554
DP(10)	60	40	1	0.0	10	3.3	48896
DP(12)	72	48	1	0.0	12	617.4	>350000
ELEV(1)	63	99	4	0.0	9	0.4	137
ELEV(2)	146	299	6	0.5	12	3.9	1061
ELEV(3)	327	783	8	5.6	15	139.0	7120
ELEV(4)	736	1939	10	157.2	>13	1215.2	43439
HART(25)	127	77	1	0.0	>5	1.0	52
HART(50)	252	152	1	0.0	>5	5.7	102
HART(75)	377	227	1	0.0	>5	15.5	152
HART(100)	502	302	1	0.0	>5	35.9	202
KEY(2)	94	92	>25	1937.9	>26	56.1	536
MMGT(3)	122	172	7	11.1	10	87.2	7702
MMGT(4)	158	232	8	687.3	>11	1874.1	66308
Q(1)	163	194	9	0.1	>17	2733.7	123596
SENT(25)	104	55	2	0.0	3	0.0	231
SENT(50)	179	80	2	0.0	3	0.0	281
SENT(75)	254	105	2	0.0	3	0.0	331
SENT(100)	329	130	2	0.0	3	0.0	381
SPD(1)	33	39	1	0.0	4	0.0	8689

Figure 3: Experiments

6 Conclusions

We present a mapping from bounded reachability and deadlock detection problems of 1-safe P/T-nets to stable model computation. Our mapping is able to handle sets of initial markings. The first experimental results indicate that stable model computation is quite a competitive approach to searching for short executions of the system leading to deadlock even from a given single initial marking and worth further study. As further work the LTL model checking and the safety LTL model checking problems look interesting. There are also alternative semantics to the two presented in this work. Experiments are needed to determine whether they are useful for bounded model checking.

References

[1] M. Anttila, H. Eriksson, J. Ikonen, R. Kujansuu, L. Ojala, and H. Tuominen. Tools and studies of formal techniques – Petri nets and temporal logic. In H. Rudin and C. West, editors, *Protocol Specification, Testing and Verification III*, pages 139–148. Elsevier Science Publishers, 1983.

[2] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99)*, pages 193–207. Springer, March 1999.

[3] J. Burch, E. Clarke, K. McMillan, D. Dill, and L.Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.

[4] E. Clarke, O. Grumberg, and D. Peled. *Model Checking.* The MIT Press, 1999.

[5] J. C. Corbett. Evaluating deadlock detection methods for concurrent software. Technical report, Department of Information and Computer Science, University of Hawaii at Manoa, 1995.

[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.

[7] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proceedings of the 5th International Conference on Logic Programming*, pages 1070–1080, Seattle, USA, August 1988. The MIT Press.

[8] K. Heljanko. Model checking the branching time temporal logic CTL. Research report A45, Helsinki University of Technology, Digital Systems Laboratory, Espoo, Finland, May 1997.

[9] 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.

[10] K. Heljanko. Using logic programs with stable model semantics to solve deadlock and reachability problems for 1-safe Petri nets. *Fundamenta Informaticae*, 37(3):247–268, 1999.

[11] T. Janhunen. Non-monotonic systems: A framework for analyzing semantics and structural properties of non-monotonic reasoning. Doctoral dissertation. Research report A49, Helsinki University of Technology, Digital Systems Laboratory, Espoo, Finland, December 1998.

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

[13] 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.

[14] T. Latvala and K. Heljanko. Coping with strong fairness. *Fundamenta Informaticae*, 43(1-4):175–193, 2000.

[15] J. McCarthy and P.J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, Edinburgh, 1969.

[16] S. Melzer and S. Römer. Deadlock checking using net unfoldings. In *Proceeding of 9th International Conference on Computer Aided Verification (CAV'97)*, pages 352–363, Haifa, Israel, Jun 1997. Springer-Verlag.

[17] R.C. Moore. Semantical considerations on nonmonotonic logic. *Artificial Intelligence*, 25:75–94, 1985.

[18] I. Niemelä. Decision procedure for autoepistemic logic. In *Proceedings of the 9th International Conference on Automated Deduction*, pages 675–684, Argonne, USA, May 1988. Springer-Verlag.

[19] I. Niemelä. On the decidability and complexity of autoepistemic reasoning. *Fundamenta Informaticae*, 17(1,2):117–155, 1992.

[20] I. Niemelä. Autoepistemic logic as a unified basis for nonmonotonic reasoning. Doctoral dissertation. Research report A24, Helsinki University of Technology, Digital Systems Laboratory, Espoo, Finland, August 1993.

[21] I. Niemelä. A decision method for nonmonotonic reasoning based on autoepistemic reasoning. *Journal of Automated Reasoning*, 14:3–42, 1995.

[22] I. Niemelä. Logic programming with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3,4):241–273, 1999.

[23] I. Niemelä and P. Simons. Efficient implementation of the well-founded and stable model semantics. In M. Maher, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 289–303, Bonn, Germany, September 1996. The MIT Press.

[24] I. Niemelä and P. Simons. Extending the Smodels system with cardinality and weight constraints. In Jack Minker, editor, *Logic-Based Artificial Intelligence*. Kluwer Academic Publishers, 2000. Accepted for publication.

[25] I. Niemelä and H. Tuominen. Helsinki Logic Machine: a system for logical expertise. Technical report B1, Helsinki University of Technology, Digital Systems Laboratory, Espoo, Finland, December 1987.

[26] J. Rintanen. Lexicographic ordering as a basis of priorities in default reasoning. Doctoral dissertation. Research report A41, Helsinki University of Technology, Digital Systems Laboratory, Espoo, Finland, December 1996.

[27] P. Simons. Extending and implementing the stable model semantics. Doctoral dissertation. Research report A58, Helsinki University of Technology, Laboratory for Theoretical Computer Science, Espoo, Finland, April 2000.

[28] H. Tuominen. Logic in Petri net analysis. Research report A5, Helsinki University of Technology, Digital Systems Laboratory, Espoo, Finland, January 1988.

[29] K. Varpaaniemi, K. Heljanko, and J. Lilius. PROD 3.2 – An advanced tool for efficient reachability analysis. In *Proceeding of the 9th International Conference on Computer Aided Verification*, pages 472–475, Haifa, Israel, June 1997. Springer-Verlag.