Answer Set Programming and Bounded Model Checking *

Keijo Heljanko and Ilkka Niemelä
Helsinki University of Technology
Dept. of Computer Science and Engineering
Laboratory for Theoretical Computer Science
P.O.Box 5400, FIN-02015 HUT, Finland
{Keijo.Heljanko, Ilkka.Niemela}@hut.fi

Abstract

In this paper bounded model checking of asynchronous concurrent systems is introduced as a promising application area for answer set programming. This is an extension of earlier work where bounded model checking has been used for verification of sequential digital circuits. As the model of asynchronous systems a generalization of communicating automata, 1-safe Petri nets, are used. A mapping from bounded reachability and deadlock detection problems of 1-safe Petri nets to stable model computation is devised. Some experimental results on solving deadlock detection problems using the mapping and the `Smodels` system are presented. They indicate that the approach is quite competitive when searching for short executions of the system leading to deadlock.

Introduction

In this paper we put forward verification and, in particular, symbolic model checking (Burch *et al.* 1992; Clarke, Grumberg, & Peled 1999) as a promising application area for answer set programming systems. In particular, we demonstrate how bounded model checking problems of asynchronous concurrent systems can be reduced to computing stable models of logic programs.

Verification of asynchronous systems is typically done by enumerating the set of reachable states of the system for all possible interleavings of atomic actions. Tools based on this approach (with various enhancements) include, e.g., the SPIN system (Holzmann 1997), which supports extended state machines communicating through FIFO queues, and the PROD tool (Varpaaniemi, Heljanko, & Lilius 1997) based on high-level Petri nets. The main problem with enumerative model checkers is the amount of memory needed to store the set of reachable states.

Symbolic model checking is widely applied especially in hardware verification. The main analysis technique is based on (ordered) binary decision diagrams (BDDs). In many cases the set of reachable states can be represented very compactly using a BDD encoding. Although the approach has been successful, there are difficulties in applying BDD-based techniques, 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 (Biere *et al.* 1999) has been proposed as a technique for overcoming the space problem by replacing BDDs with 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 a sequential 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. Hence, bounded model checking provides directly interesting and practically relevant benchmarks for any answer set programming system capable of handling propositional satisfiability problems. 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. In order to illustrate the approach we use a simple basic model of asynchronous systems. We employ Petri nets and, in particular, focus on 1-safe Place/Transition nets (P/T-nets) as an interesting generalization of communicating automata (Desel & Reisig 1998). It turns out that bounded model checking for 1-safe P/T-nets is closely related to planning and techniques used in, e.g., SAT planning could be employed. 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 (Niemelä 1999).

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 an extension of

*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, and Emil Aaltonen Foundation are gratefully acknowledged.

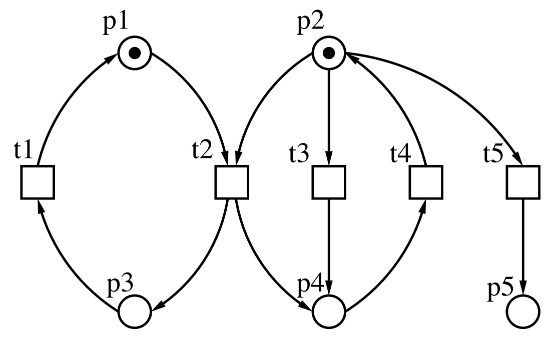

Figure 1: Running Example: A 1-safe P/T-net

the stable model semantics which we employ in the following section to achieve a compact encoding of bounded model checking using logic programs. We discuss mappings to alternative ASP formalism, present some experimental results, and end with some concluding remarks.

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*. The places are represented in graphical notation by circles, transitions by squares, and the *flow relation* F with arcs.

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 $\{y \in P \cup T \mid F(y, x) = 1\}$. In our running example, e.g., ${}^\bullet t2 = \{p1, p2\}$. The *postset* of a node x, denoted by $x^\bullet$, is the set $\{y \in P \cup T \mid F(x, y) = 1\}$. Again in our running example $p2^\bullet = \{t2, t3, t5\}$.

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 is graphically denoted by a distribution of tokens on the places of the net. In our running example in Figure 1 the net has the initial marking $M_0 = \{p1, p2\}$.

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 $\leq n$ 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. Notice that 1-safeness is a semantic property. However, it can be guaranteed by construction, as is usually done in modeling. 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. (It is easy to prove by using the 1-safeness of the P/T-net Σ that all possible orders of transitions in a step S are enabled in M, and that they all lead to the same final marking M'.) 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 $M' \xrightarrow{\{t1,t4\}} M_0$. Notice also that for any enabled transition, the singleton set containing only that transition is always (trivially) a step.

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. Note that if a marking is reachable in n transitions in the interleaving semantics, it is also reachable in n steps in the step semantics. However, the converse does not necessarily hold. We have, however, 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.*

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?*

The reachability and deadlock problems for 1-safe Petri nets are PSPACE-complete (Jones, Landweber, & Lien 1977; Esparza 1998).

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?*

It is straightforward to prove that the bounded versions of the problems are NP-complete when the bound n is given in unary encoding. We can think about the bounded versions of the problems as approximations of the original problem which become increasingly better as the bound n increases.

The main motivation for using the bounded version is that if we find a solution, then the original problem has also that same solution. If not, we can increase the bound, and our approximation becomes better. Notice that if we set the bound to be $n = (2^{|P|} - 1)$, the bounded and non-bounded versions are guaranteed to be equivalent for both problems and semantics. This is easy to see, as $2^{|P|}$ is the upper bound on the number of 1-safe markings. Using such a bound is, however, not practical for most systems having hundreds or thousands of places. For most net systems a smaller bound suffices for completeness.

The concurrency between transitions in the step semantics often makes it possible to reach states using a smaller bound than for the interleaving semantics. The choice of semantics can have quite significant effects on the performance of the bounded model checking in practice (see Experiments).

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 (Biere *et al.* 1999) 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 the bounded model checking results are usually not conclusive if a solution is not found. Therefore, bounded model checking is at its best in "bug hunting", and not as easily applicable in verifying systems to be correct.

Stable model semantics

In this section we introduce logic programs and the stable model semantics originally presented in (Gelfond & Lifschitz 1988) for normal logic programs of the form

$$a \leftarrow b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n \,. \tag{1}$$

Recently, this approach has been extended to handle new kinds of constructs such as cardinality and weight constraints (Niemelä, Simons, & Soininen 1999; Niemelä & Simons 2000). In this work we employ rules with cardinality constraints in order to obtain a succinct and simple encoding of model checking problems. The rest of the section reviews the stable model semantics for such rules.

A *cardinality constraint* is an expression of the form

$$l \; \{a_1, \ldots, a_n, \text{not } b_1, \ldots, \text{not } b_m\} \; u \tag{2}$$

where l and u are two integers giving the *lower* and *upper bound* of the constraint, respectively. For a cardinality constraint C (2), we denote by $\text{lit}(C)$ the corresponding set of literals $\{a_1, \ldots, a_n, \text{not } b_1, \ldots, \text{not } b_m\}$. The idea is that such a constraint is satisfied by a model for which the cardinality of the subset of the literals satisfied by the model is between the integers l and u inclusive. Either of the bounds can be omitted in which case a missing lower bound is be taken as 0 and upper bound as ∞.

Cardinality constraint rules are of the form $C_0 \leftarrow C_1, \ldots, C_n$ where each C_i is a cardinality constraint. They are a generalization of normal rules, i.e., a literal p can be seen as a shorthand for a cardinality constraint $1\{p\}$. For instance, a rule

$$\{a_1, a_2, a_3\} \leftarrow 1 \; \{\text{not } b_1, \text{not } b_2\} \; , 1 \; \{c_1, c_2, c_3\} \; 2, d$$

says that if at least one of b_1, b_2 is missing from a stable model, at least 1 but at most 2 from $\{c_1, c_2, c_3\}$ are included, and d is included, then some subset of $\{a_1, a_2, a_3\}$ is contained the model. Note that the empty set is also a possible choice for the subset.

The semantics for cardinality constraint rules is a generalization of the stable model semantics for normal logic programs and is given in terms of models that are sets of atoms. Given a model S (a set of atoms) we use the notation $S \models a$ iff $a \in S$ and $S \models \text{not } a$ iff $a \notin S$.

Definition 5 *A set of atoms S* satisfies *a cardinality constraint C of the form (2) ($S \models C$) iff $l \leq \text{W}(C, S) \leq u$ where*

$$\text{W}(C, S) = |\{p \in \text{lit}(C) : S \models p\}|$$

is the number of literals in C satisfied by S.

A rule $C_0 \leftarrow C_1, \ldots, C_n$ is satisfied by S ($S \models C_0 \leftarrow C_1, \ldots, C_n$) iff S satisfies C_0 whenever it satisfies each of $C_1, \ldots, C_n$.

We also allow *integrity constraints*, i.e., rules without the *head* constraint C_0, which are satisfied if at least one of the *body* constraints $C_1, \ldots, C_n$ is not.

The idea is to define a stable model of a set of rules as a set of atoms that satisfies the rules and is justified by them. Justifiability is captured by generalizing the concept of a *reduct* used for normal rules (Gelfond & Lifschitz 1988).

The reduct C^S of a constraint C of the form (2) w.r.t. a set of atoms S is the constraint

$$l' \; \{a_1, \ldots, a_n\} \tag{3}$$

where $l' = l - |\{\text{not } b \in \text{lit}(C) : S \models \text{not } b\}|$. Hence, in the reduct all negative literals and the upper bound are removed and the lower bound is decreased by the number of negative literals satisfied by S to account for the contribution of the negative literals towards satisfying the lower bound. For example, for a set $S = \{q\}$ and a constraint C

$$3 \; \{\text{not } q, \text{not } r, p\} \; 4$$

the reduct C^S is $2 \; \{p\}$.

The reduct Π^S for a program Π w.r.t. a set of atoms S is a set of rules which contains a rule r' with an atom p as the head if $p \in S$ and there is a rule $r \in \Pi$ such that p appears in the head and the upper bounds of the constraints in the body of r are satisfied by S. The body of r' is obtained by taking the reduct of the constraints in the body of r. Formally the reduct is defined as follows.

Definition 6 *Let Π be a ground program and S a set of ground atoms. The reduct Π^S of Π w.r.t. S is defined by*

$$\begin{aligned}\Pi^S \quad = \quad & \{p \leftarrow C_1^S, \ldots, C_n^S : C_0 \leftarrow C_1, \ldots, C_n \in \Pi, \\ & p \in \mathrm{lit}(C_0) \cap S \textit{ and for all } i = 1, \ldots, n, \\ & \textit{for the constraint } C_i \textit{ of the form} \\ & l\ \{a_1, \ldots, \textit{not}\ b_m\}\ u\ , \mathrm{W}(C_i, S) \leq u\}\end{aligned}$$

The role of the reduct is to provide the possible justifications for the atoms in S. Each atom in a stable model is justified by the program in the sense that it is in the closure of the reduct. The reduct is a set of rules of the form

$$h \leftarrow C_1, \ldots, C_n \tag{4}$$

where h is a ground atom and each constraint C_i contains only positive literals and has only a lower bound condition. The closure $\mathrm{cl}(\Pi)$ of a reduct Π is defined as the unique smallest set of atoms satisfying Π. The uniqueness is implied by the monotonicity of reduct rules, i.e., if the body of a rule is satisfied by a model S, then it is satisfied by any superset of S.

Definition 7 *A set of ground atoms S is a* stable model *of a program Π iff $S \models \Pi$ and $S = \mathrm{cl}(\Pi^S)$.*

Example 1 *Consider a program Π*

$$1\ \{a_1, a_2, a_3\}\ 1 \leftarrow$$

Observe that a stable model of a program Π must be a subset of the atoms appearing in the heads of the rules in Π because other atoms cannot appear in the closure of a reduct.

The empty set is not a stable model because $\emptyset \not\models 1\ \{a_1, a_2, a_3\}\ 1$ and similarly for every subset having more than one of the atoms. However, $\{a_1\}$ is a stable model of Π because it satisfies the rule and the reduct $\Pi^{\{a_1\}} = \{a_1 \leftarrow\}$ has $\{a_1\}$ as its closure. In fact, Π has three stable models $\{a_1\}$, $\{a_2\}$, and $\{a_3\}$ as one would expect.

A program

$$\{a\}\ \leftarrow \tag{5}$$

has two stable models $\{\}$ and $\{a\}$ demonstrating that stable models are not necessarily subset minimal.

For the model checking applications in this paper two features of cardinality constraints are important. One is their ability to encode choices over subsets with rules of the type (5). These kinds of choices can be encoded using normal rules only by introducing new extra atoms. The second feature involves a conflict with two atoms out of a large set of atoms, i.e., a rule of the form

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

which disallows any stable model contain at least two atoms from $\{a_1, \ldots, a_n\}$. There seems to be no simple compact encoding of such a condition using normal rules. The `Smodels` system (`http://www.tcs.hut.fi/Software/smodels/`), which provides an implementation for cardinality constraint rules, includes primitives supporting directly such constraints.

From bounded model checking to answer set programming

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 of cardinality constraint rules. We end the section by discussing how a similar mapping could be done using normal programs or propositional logic.

Consider a net $N = \langle P, T, F \rangle$ and a step bound n. We construct a logic program $\Pi_{\mathrm{A}}(N, n)$, which captures the possible executions of N up to n steps, as follows.

- For each place $p \in P$, include a choice rule
$$\{p(0)\} \leftarrow \tag{6}$$
- For each transition $t \in T$, and for all $i = 0, 1, \ldots, n-1$, include a rule
$$\{t(i)\} \leftarrow p_1(i), \ldots, p_l(i) \tag{7}$$
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
$$p(i+1) \leftarrow 1\{t_1(i), \ldots, t_k(i)\} \tag{8}$$
where $\{t_1, \ldots, t_k\}$ is the preset of p. This says that p holds in the next step if at least one of its preset transitions is in the current step.
- For each place $p \in P$, and for all $i = 0, 1, \ldots, n-1$, include a rule
$$\leftarrow 2\{t_1(i), \ldots, t_l(i)\} \tag{9}$$
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 $i = 0, 1, \ldots, n-1$,
$$p(i+1) \leftarrow p(i), \mathrm{not}\ t_1(i), \ldots, \mathrm{not}\ t_l(i) \tag{10}$$
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 the net N in Figure 1. The program $\Pi_{\mathrm{A}}(N, n)$ is given in Figure 2. In the program $\Pi_{\mathrm{A}}(N, n)$ the initial marking is not constrained but additional conditions on markings can be stated using rules. For example, stable models not satisfying a marking M at step i can be eliminate with rules

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

Example 2 *For the initial marking M_0 of our running example, the set $\Pi_{\mathrm{M}}(M_0, 0)$ is*

$$\begin{array}{lll} \leftarrow \textit{not}\ p1(0) & \leftarrow p3(0) & \leftarrow p5(0) \\ \leftarrow \textit{not}\ p2(0) & \leftarrow p4(0) & \end{array}$$

Now the stable models of the program $\Pi_{\mathrm{M}}(M_0, 0) \cup \Pi_{\mathrm{A}}(N, n)$ capture the markings reachable in n steps from

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

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

M_0. For the bound $n = 1$, the program has four stable models corresponding to the four possible steps:

$$
\begin{array}{l}
\{p1(0), p2(0), p1(1), p2(1)\} \\
\{p1(0), p2(0), t2(0), p3(1), p4(1)\} \\
\{p1(0), p2(0), t3(0), p1(1), p4(1)\} \\
\{p1(0), p2(0), t5(0), p1(1), p5(1)\}
\end{array}
$$

For example, the first one corresponds to the empty step and the second to a step where transition $t2$ occurs.

Any Boolean combination C of marking conditions can be captured using a similar set of rules $\Pi_{\mathrm{M}}(C, i)$. For example, for eliminating stable models not satisfying a condition C at step i requiring that $M(p_1) = 1$ and ($M(p_2) = 0$ or $M(p_3) = 1$), it is sufficient to use rules $\Pi_{\mathrm{M}}(C, i)$:

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

Now given a condition C_0 capturing initial markings for which a net N is 1-safe, the stable models of $\Pi_{\mathrm{M}}(C_0, 0) \cup \Pi_{\mathrm{A}}(N, n)$ correspond to all executions of N up to n steps from any initial marking satisfying C_0. Hence, our approach can solve a reachability problem for a set of initial markings given by a condition C_0 where the markings to be reached are specified by another condition C.

Theorem 2 *Let $N = \langle P, T, F\rangle$ be a 1-safe P/T-net for all initial markings satisfying a condition C_0. Net N has an initial marking satisfying C_0 such that a marking satisfying a condition C is reachable in at most n steps iff $\Pi_{\mathrm{M}}(C_0, 0) \cup \Pi_{\mathrm{A}}(N, n) \cup \Pi_{\mathrm{M}}(C, n)$ 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

$$\leftarrow p_1(n), \ldots, p_l(n) \tag{11}$$

where $\{p_1, \ldots, p_l\}$ is the preset of t.

For our running example, the 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 a condition C_0. Net N has an initial marking satisfying C_0 such that a deadlock is reachable in at most n steps iff $\Pi_{\mathrm{M}}(C_0, 0) \cup \Pi_{\mathrm{A}}(N, n) \cup \Pi_{\mathrm{D}}(N, n)$ 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 all the problems by adding a set of rules $\Pi_{\mathrm{I}}(N, n)$ to the step version of the problem. The set $\Pi_{\mathrm{I}}(N, n)$ includes for each time step $0 \le i \le n-1$ a rule

$$\leftarrow 2\{t_1(i), \ldots, t_m(i)\} \tag{12}$$

where $\{t_1, \ldots, t_m\}$ is the set of all transitions. These rules 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 using a translation given above. Then the program $\Pi_{\mathrm{S}}(N, n) \cup \Pi_{\mathrm{I}}(N, n)$ solves the same problem in the interleaving semantics.*

In (Biere *et al.* 1999) 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. One of the 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 (Kupferman & Vardi 1999), as a simpler translation for that LTL subset is possible.

Mappings to other ASP formalisms

Normal programs The mappings described above could be done to normal logic programs. In fact, only rules (6), (7), (8), (9), and (12) are not normal ones. The first three are simple to handle. For example, (7) can be replaced by two normal rules

$$
\begin{array}{l}
t(i) \leftarrow \text{not } t'(i), p_1(i), \ldots, p_l(i) \\
t'(i) \leftarrow \text{not } t(i)
\end{array}
\tag{13}
$$

where a new atom $t'(i)$ is introduced and (8) with k rules

$$
\begin{array}{l}
p(i+1) \leftarrow t_1(i) \\
\ldots \\
p(i+1) \leftarrow t_k(i).
\end{array}
$$

However, the case of (9) and (12) is more challenging and there seems to be no simple way of encoding such conditions using only a linear number of normal rules. Hence, the mappings can be done using normal rules but because of conditions such as (9), the number of rules for each step in the resulting program is not linear in the size of the net as is the case for the mapping to cardinality constraint rules.

Propositional satisfiability The mapping from P/T-nets to propositional satisfiability is also fairly straightforward to construct. For example, one can use the mapping to normal programs discussed above as the basis. The program is acyclic except for rules (13). Hence, one can employ Clark's completion and an extension of Fages' theorem (Fages 1994) discussed in (Babovich, Erdem, & Lifschitz 2000). However, the size of the set of the resulting propositional formulae for each step is not linear w.r.t. the size of the -net because of the difficulties in encoding cardinality conditions of the form (9) compactly using propositional formulae.

A further complication is caused by the fact that most efficient satisfiability checkers require that the input formulae are transformed to conjunctive normal form (CNF). It is non-trivial to tune the CNF transformation such that the checkers have a reasonable performance. The basic problem is that an equivalent CNF formula can be exponentially bigger than the original formula. This explosion is typically avoided by introducing new atoms corresponding to subformulae but the new atoms can increase the search space of the checker exponentially.

Experiments

We have implemented the translation of Theorem 2 from the bounded model checking problem to the problem of finding a stable model. Also the deadlock checking part $\Pi_{\mathrm{D}}(N, n)$ and the interleaving semantics part $\Pi_{\mathrm{I}}(N, n)$ can be optionally added. The translation was implemented in C++ in quite a straightforward manner with only two simple optimizations included:

- 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. (Duplicates can appear in the conflict (9) and liveness (11) rules.)

As benchmarks we use a set of deadlock checking benchmarks collected by Corbett (1995), where more detailed information about them can be found. They have been converted from communicating state machines to 1-safe P/T-nets by Melzer and Römer (1997). The models were picked from those which have a deadlock. For each model and both semantics we incremented the used bound until a deadlock was found. We report the time for `smodels` to find the first stable model using this bound. 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 reported bound.

The experimental results can be found in Fig. 3. The columns of the table are the following:

- Problem: The problem name with the size of the instance in parenthesis.

Problem	$\|P\|$	$\|T\|$	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

- $|P|$: Number of places in the original net.
- $|T|$: Number of 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 reported 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. (The time needed for creating the smodels input was quite small, and therefore omitted.)

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 it 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 are promising, in particular, for small bounds and the step semantics. However, we need to get a better understanding of the behavior of the bounded model checking approach by doing more experiments.

Conclusions

We introduce bounded model checking of asynchronous concurrent systems modeled by 1-safe P/T-nets as an interesting application area for answer set programming. We present a mapping from bounded reachability and deadlock detection problems of 1-safe P/T-nets to stable model computation. The first experimental results indicate that stable model computation is a quite competitive approach to searching for short executions of the system leading to deadlock and worth further study.

In our approach it is possible to do model checking for a set of initial markings at once. This is usually difficult to achieve in current enumerative model checkers and often leads to state space explosion. All our benchmark experiments used only a single initial state, as they were originally designed for a tool which does not support this feature. Thus more experimental work is needed on this aspect of the translation. The bounded model checking translation can also be seen more goal directed than the explicit state version, as the constraints based on the final state of the system can guide the search.

The net unfolding method (see (Heljanko 1999; Melzer & Römer 1997) and further references there) is another symbolic model checking approach for asynchronous systems, where answer set programming has been employed. Relating this approach to bounded model checking would be interesting. 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

Babovich, Y.; Erdem, E.; and Lifschitz, V. 2000. Fages' theorem and answer set programming. In *Proceedings of the 8th International Workshop on Non-Monotonic Reasoning*. cs.AI/0003042.

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

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

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

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

Desel, J., and Reisig, W. 1998. Place/Transition Petri nets. In *Lectures on Petri Nets I: Basic Models*. Springer-Verlag. 122–173.

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

Fages, F. 1994. Consistency of Clark's completion and existence of stable models. *Journal of Methods of Logic in Computer Science* 1:51–60.

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

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

Holzmann, G. 1997. The model checker SPIN. *IEEE Transactions on Software Engineering* 23(5):279–295.

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

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

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

Niemelä, I., and Simons, P. 2000. Extending the Smodels system with cardinality and weight constraints. In Minker, J., ed., *Logic-Based Artificial Intelligence*. Kluwer Academic Publishers. 491–521.

Niemelä, I.; Simons, P.; and Soininen, T. 1999. Stable model semantics of weight constraint rules. In *Proceedings of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning*, 317–331. El Paso, Texas, USA: Springer-Verlag.

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

Varpaaniemi, K.; Heljanko, K.; and Lilius, J. 1997. PROD 3.2 - An advanced tool for efficient reachability analysis. In *Proceedings of the 9th International Conference on Computer Aided Verification (CAV'97)*, 472–475. Haifa, Israel: Springer-Verlag.