Fundamenta Informaticae 37 (1999) 247–268
IOS Press

Using Logic Programs with Stable Model Semantics to Solve Deadlock and Reachability Problems for 1-Safe Petri Nets*

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

Abstract. McMillan has presented a deadlock detection method for Petri nets based on finite complete prefixes (i.e. net unfoldings). The approach transforms the PSPACE-complete deadlock detection problem for a 1-safe Petri net into a potentially exponentially larger NP-complete problem of deadlock detection for a finite complete prefix. McMillan devised a branch-and-bound algorithm for deadlock detection in prefixes. Recently, Melzer and Römer have presented another approach, which is based on solving mixed integer programming problems. In this work it is shown that instead of using mixed integer programming, a constraint-based logic programming framework can be employed, and a linear-size translation from deadlock detection in prefixes into the problem of finding a stable model of a logic program is presented. As a side result also such a translation for solving the reachability problem is devised. Correctness proofs of both the translations are presented. Experimental results are given from an implementation combining the prefix generator of the PEP-tool, the translation, and an implementation of a constraint-based logic programming framework, the `smodels` system. The experiments show the proposed approach to be quite competitive, when compared to the approaches of McMillan and Melzer/Römer.

Keywords: Verification, Petri nets, logic programs, deadlock checking, reachability

*This work is an extended version of [10].

†Address for correspondence: Laboratory for Theoretical Computer Science, Helsinki University of Technology, P.O. Box 5400, 02015 HUT, Finland

1. Introduction

Petri nets are a widely used model for analyzing concurrent and distributed systems. Often such a system must exhibit reactive, non-terminating behavior, and one of the key analysis problems is that of deadlock-freedom: Do all reachable global states of the system (markings of the net) enable some action (net transition)? In this work we study this problem for a subclass of Petri nets, the 1-safe Petri nets, which are capable of modelling finite state systems. For 1-safe Petri nets the deadlock detection problem is PSPACE-complete in the size of the net [4], however, restricted subclasses of 1-safe Petri nets exist for which this problem is NP-complete [12, 13]. McMillan has presented a deadlock detection method for Petri nets based on finite complete prefixes (i.e. net unfoldings) [12, 13]. The basic idea is to transform the PSPACE-complete deadlock detection problem for a 1-safe Petri net into a potentially exponentially larger NP-complete problem. This translation creates a finite complete prefix, which is an acyclic 1-safe Petri net of a restricted form. Experimental results show that the blowup of the transformation can in many cases be avoided [5, 12, 13, 14].

In this work we address the NP-complete deadlock detection problem for finite complete prefixes. McMillan originally suggested a branch-and-bound algorithm for solving this problem. Recently, Melzer and Römer have presented another algorithm which is based on solving mixed integer programming problems generated from prefixes [14]. Their approach seems to be faster than McMillan's on examples in which a large percentage of the events of the prefix are so called cut-off events. However, if this assumption does not hold, the run times are generally slower than those of the McMillan's algorithm [14].

In this work we study an approach that is similar to that of Melzer and Römer in the way of being capable of handling cases with a large percentage of cut-off events but with more competitive performance. Instead of mixed integer programming our approach is based on a constraint-based logic programming framework [15, 16, 17]. We translate the deadlock detection problem into the problem of finding a stable model of a logic program. As a side result we also obtain such a translation for checking the reachability problem, which is also NP-complete in the size of the prefix [4]. The main contribution of this work also includes the detailed correctness proofs of the translations. For the deadlock detection problem we present experimental results, and find our approach competitive with the two previous approaches.

The rest of the paper is divided as follows. First we present Petri net notations used in the paper. In Sect. 3 we will introduce the rule-based constraint programming framework. Section 4 contains the main results of this work, linear-size translations from deadlock and reachability property checking into the problem of finding a stable model of a logic program, and their correctness proofs. In Sect. 5 we present experimental results from our implementation. In Sect. 6 we conclude and discuss directions for future research.

2. Petri Net Definitions

First we define basic Petri net notations. Next we introduce *occurrence nets*, which are 1-safe Petri nets of a restricted form. Then *branching processes* are given as a way of describing partial order semantics for Petri nets. Last but not least we define *finite complete prefixes* as a way of giving a finite representation of this partial order behavior. We follow mainly the notation of [5, 14].

2.1. Petri Nets

A triple $\langle S, T, F\rangle$ is a *net* if $S \cap T = \emptyset$ and $F \subseteq (S \times T) \cup (T \times S)$. The elements of S are called *places*, and the elements of T *transitions*. Places and transitions are also called *nodes*. We identify F with its characteristic function on the set $(S \times T) \cup (T \times S)$. The *preset* of a node x, denoted by ${}^\bullet x$, is the set $\{y \in S \cup T \mid F(y, x) = 1\}$. The *postset* of a node x, denoted by $x^\bullet$, is the set $\{y \in S \cup T \mid F(x, y) = 1\}$. Their generalizations on sets of nodes $X \subseteq S \cup T$ are defined as ${}^\bullet X = \bigcup_{x \in X} {}^\bullet x$, and $X^\bullet = \bigcup_{x \in X} x^\bullet$ respectively.

A *marking* of a net $\langle S, T, F\rangle$ is a mapping $S \mapsto \mathbb{N}$. A marking M is identified with the multi-set which contains $M(s)$ copies of s for every $s \in S$. A 4-tuple $\Sigma = \langle S, T, F, M_0\rangle$ is a *net system* if $\langle S, T, F\rangle$ is a net and M_0 is a marking of $\langle S, T, F\rangle$. A marking M enables a transition t if $\forall s \in S : F(s, t) \leq M(s)$. If t is enabled, it can *occur* leading to a new marking (denoted $M \xrightarrow{t} M'$), where M' is defined by $\forall s \in S : M'(s) = M(s) - F(s, t) + F(t, s)$. A marking M is a *deadlock marking* iff no transition t is enabled by M. A marking M_n is *reachable* in Σ iff there exist a sequence of transitions $t_1, t_2, \ldots, t_n$ and markings $M_1, M_2, \ldots, M_{n-1}$ such that: $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \ldots M_{n-1} \xrightarrow{t_n} M_n$. A reachable marking is 1-safe if $\forall s \in S : M(s) \leq 1$. A net system Σ is 1-safe if all its reachable markings are 1-safe. In this work we will restrict ourselves to the set of net systems which are 1-safe, have a finite number of places and transitions, and also in which each transition $t \in T$ has both nonempty pre- and postsets.

2.2. Occurrence Nets

We use $\leq_F$ to denote the reflexive transitive closure of F. Let $\langle S, T, F\rangle$ be a net and let $x_1, x_2 \in S \cup T$. The nodes x_1 and x_2 are in *conflict*, denoted by $x_1 \# x_2$, if there exist $t_1, t_2 \in T$ such that $t_1 \neq t_2$, ${}^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset$, $t_1 \leq_F x_1$, and $t_2 \leq_F x_2$. An occurrence net is a net $N = \langle B, E, F\rangle$ such that:

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

The elements of B and E are called *conditions* and *events*, respectively. The set $Min(N)$ denotes the set of minimal elements of the transitive closure of F. A *configuration* C of an occurrence net is a set of events satisfying:

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

The co-set of a configuration is called a cut: $Cut(C) = (Min(N) \cup C^\bullet) \setminus {}^\bullet C$. A configuration C is a *deadlock configuration* iff the set $Cut(C)$ does not enable any event $e \in E$.

2.3. Branching Processes

Branching processes are "unfoldings" of net systems and were introduced by Engelfriet [3]. Let $N_1 = \langle S_1, T_1, F_1 \rangle$ and $N_2 = \langle S_2, T_2, F_2 \rangle$ be two nets. A *homomorphism* is a mapping $S_1 \cup T_1 \mapsto S_2 \cup T_2$ such that: $h(S_1) \subseteq S_2 \wedge h(T_1) \subseteq T_2$, and for all $t \in T_1$, the restriction of h to ${}^\bullet t$ is a bijection between ${}^\bullet t$ and ${}^\bullet h(t)$, and similarly for $t^\bullet$ and $h(t)^\bullet$. A *branching process* of a net system Σ is a tuple $\beta = \langle N', h \rangle$, where N' is a occurrence net, and h is a homomorphism from N' to $\langle S, T, F \rangle$ such that: the restriction of h to $Min(N')$ is a bijection between $Min(N')$ and M_0, and $\forall e_1, e_2 \in E$, if ${}^\bullet e_1 = {}^\bullet e_2 \wedge h(e_1) = h(e_2)$ then $e_1 = e_2$. The set of places associated with a configuration C of β is denoted by $Mark(C) = h(Cut(C))$.

2.4. Finite Complete Prefixes

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

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

Algorithms to obtain a finite complete prefix β given a 1-safe net system Σ are presented in e.g. [5, 12, 13]. The algorithms will mark some events of the prefix β as special *cut-off events*, which we denote by the set $CutOffs(\beta) \subseteq E$. The intuition behind cutoff events is that for each cut-off event e there already exists another event e' in the prefix. The markings reachable after executing e can also be reached after executing e', and thus the markings after e need not to be considered any further. We direct the reader interested in the approach to [5, 12, 13, 14].

3. Rule-Based Constraint Programming

We will use normal logic programs with stable model semantics [7] as the underlying formalism into which the deadlock and reachability problems for 1-safe Petri nets are translated. This section is to a large extent based on [17].

The stable model semantics is one of the main declarative semantics for normal logic programs. However, here we use logic programming in a way that is different from the typical PROLOG style paradigm, which is based on the idea of evaluating a given query. Instead, we employ logic programs as a *constraint programming framework* [15], where stable models are

the solutions of the program rules seen as constraints. We consider normal logic programs that consist of rules of the form

$$\mathtt{h} \leftarrow \mathtt{a}_1, \ldots, \mathtt{a}_n, \mathit{not}\,(\mathtt{b}_1), \ldots, \mathit{not}\,(\mathtt{b}_m) \tag{1}$$

where $\mathtt{a}_1, \ldots, \mathtt{a}_n, \mathtt{b}_1, \ldots, \mathtt{b}_m$ and $\mathtt{h}$ are propositional atoms. Such a rule can be seen as a constraint saying that if atoms $\mathtt{a}_1, \ldots, \mathtt{a}_n$ are in a model and atoms $\mathtt{b}_1, \ldots, \mathtt{b}_m$ are not in a model, then the atom $\mathtt{h}$ is in a model. The stable model semantics also enforces minimality and groundedness of models. This makes many combinatorial problems easily and succinctly describable using logic programming with stable model semantics.

We will demonstrate the basic behavior of the semantics using programs P1-P4:

P1: $\mathtt{a} \leftarrow \mathit{not}\,(\mathtt{b})$, $\mathtt{b} \leftarrow \mathit{not}\,(\mathtt{a})$ P2: $\mathtt{a} \leftarrow \mathtt{a}$ P3: $\mathtt{a} \leftarrow \mathit{not}\,(\mathtt{a})$ P4: $\mathtt{a} \leftarrow \mathtt{c}, \mathit{not}\,(\mathtt{b})$, $\mathtt{b} \leftarrow \mathit{not}\,(\mathtt{a})$

Program P1 has two stable models: $\{\mathtt{a}\}$ and $\{\mathtt{b}\}$. The property of this program is that we may freely make negative assumptions as long as we do not bump into any contradictions. For example, we may assume $\mathit{not}\,(\mathtt{b})$ in order to deduce the stable model $\{\mathtt{a}\}$. Program P2 has the empty set as its unique stable model. This exposes the fact that we can't use positive assumptions to deduce what is to be included in a model. Program P3 is an example of a program which has no stable models. If we assume $\mathit{not}\,(\mathtt{a})$, then we will deduce $\mathtt{a}$, which will contradict with our assumption $\mathit{not}\,(\mathtt{a})$. Program P4 has one stable model $\{\mathtt{b}\}$. If we assume $\mathit{not}\,(\mathtt{a})$ then we will deduce $\mathtt{b}$. If we assume $\mathit{not}\,(\mathtt{b})$ then we can't deduce $\mathtt{a}$, because $\mathtt{c}$ can't be deduced from our assumptions.

The stable model semantics for a normal logic program P is defined as follows [7]. The reduct P^A of P with respect to the set of atoms A is obtained (i) by deleting each rule in P that has a not-atom $\mathit{not}\,(\mathtt{x})$ in its body such that $\mathtt{x} \in A$ and (ii) by deleting all not-atoms in the remaining rules. A set of atoms A is a stable model of P if and only if A is the deductive closure of P^A when the rules in P^A are seen as inference rules.

A non-deterministic way of constructing stable models is to guess which assumptions (not-atoms of the program) to use, and then check using the deductive closure (in linear time) whether the resulting model agrees with the assumptions. The problem of determining the existence of a stable model is in fact NP-complete [11].

Next we give rest of the stable model semantics definitions, which are needed in the proofs.

Definition 3.1. Let A be a set of atoms, we define $\mathit{not}\,(A) = \{\mathit{not}\,(\mathtt{a}) \mid \mathtt{a} \in A\}$.

For a set of atoms and not-atoms B we denote the atoms in B by B^+ and the set of not-atoms by B^-. Atoms and not-atoms are also called *literals*. We denote with $\mathit{Atoms}(P)$ the set of all propositional atoms which appear in the logic program P as literals. We use the notation $\overline{\Delta}$ to denote the set $\mathit{Atoms}(P) \setminus \Delta$.

Definition 3.2. The deductive closure of a set of rules P and a set of literals B is denoted by $\mathit{Dcl}\,(P, B)$, where $\mathit{Dcl}\,(P, B)$ is the smallest set of atoms that contains B^+ and is closed under

$R(P,B)$ when

$$R(P,B) = \{\mathtt{h} \leftarrow \mathtt{a_1}, \ldots, \mathtt{a_n}, not(\mathtt{b_1}), \ldots, not(\mathtt{b_m}) \in P \text{ and } not(\mathtt{b_i}) \in B^-, \text{ for } i = 1, \ldots, m\}$$

is seen as a set of inference rules.

The deductive closure gives us a fixpoint characterization of the stable models.

Proposition 3.1. *The set Δ is a stable model of a set of rules P iff $\Delta = Dcl(P, not(\overline{\Delta}))$.*

Proof:
Note that the reduct $P^{\Delta} = R(P, not(\overline{\Delta}))$. □

3.1. The tool `smodels`

There is a tool, the `smodels` system [16, 17], which provides an implementation of logic programs as a rule-based constraint programming framework. It finds stable models of a logic program, and can also tell when the program has no stable models. The implementation is based on backtracking search techninique similar to the Davis Putnam method (see e.g. [6]), and it uses a generalization of the well-founded semantics [20] to approximate the stable models and to prune the search space. The `smodels` implementation needs space linear in the size of the input program [17]. The `smodels` seems to be the most efficient implementation of the stable model semantics currently available and it has been applied successfully in a number of areas including planning and propositional satisfiability checking, see, e.g. [17].

The stable model semantics is defined using rules of the form (1). The `smodels` 2 handles extended rule types [18], which can be seen as succinct encodings of sets of basic rules. One of the rule types is a rule of the form: $\mathtt{h} \leftarrow 2\{\mathtt{a_1}, \ldots, \mathtt{a_n}\}$. The semantics of this rule is that if two or more atoms from the set $\mathtt{a_1}, \ldots, \mathtt{a_n}$ belong to the model, then also the atom `h` will be in the model. It is easy to see that this rule can be encoded by using $\frac{N^2-N}{2}$ basic rules of the form: $\mathtt{h} \leftarrow \mathtt{a_i}, \mathtt{a_j}$. Using an extended rule instead of the corresponding basic rule encoding was necessary to achieve a linear-size translation of the two problems at hand.

We also use the so called *integrity rules* in the programs. They are rules with no head, i.e. of the form: $\leftarrow \mathtt{a_1}, \ldots, \mathtt{a_n}, not(\mathtt{b_1}), \ldots, not(\mathtt{b_m})$. The semantics is given by the following: A new atom `f` is introduced to the program, and the integrity rule is replaced by: $\mathtt{f} \leftarrow \mathtt{a_1}, \ldots, \mathtt{a_n}, not(\mathtt{b_1}), \ldots, not(\mathtt{b_m}), not(\mathtt{f})$. It is easy to see that any set of atoms, such that $\mathtt{a_1}, \ldots, \mathtt{a_n}$ are in a model and atoms $\mathtt{b_1}, \ldots, \mathtt{b_m}$ are not in a model, is not a stable model. It is also easy to see that adding one integrity rule doesn't create any new stable models, and neither does adding any set of integrity rules. The last extended rule we use is of the form: $\{\mathtt{h}\} \leftarrow \mathtt{a_1}, \ldots, \mathtt{a_n}$. The semantics is the following: A new atom $\mathtt{h'}$ is introduced to the program, and the rule is replaced by two rules: $\mathtt{h} \leftarrow \mathtt{a_1}, \ldots, \mathtt{a_n}, not(\mathtt{h'})$, and $\mathtt{h'} \leftarrow not(\mathtt{h})$. The atom $\mathtt{h'}$ is removed from any stable models it appears in, and the rest of the model gives the semantics for the extended rule.

4. Translating Deadlock and Reachability Property Checking into Logic Programs

In this section we present the translations of deadlock and reachability properties into logic programs with stable model semantics. For the deadlock property the main result can be seen as a rephrasing of the Theorem 4 of [14], where mixed integer programming has been replaced by the rule-based constraint programming framework. For the reachability property we give another translation. In this work we assume that the set of events of a finite complete prefix is non-empty. If it is empty, the corresponding net system would have no events enabled in the initial state, and then the deadlock and reachability properties can be trivially solved by looking at the initial state only.

Now we are ready to define our translation from the finite complete prefixes into logic programs with stable model semantics. The basic part of our translation is given next. It translates the notion of a configuration of a finite complete prefix into the problem of finding a stable model of a logic program. The definitions will be followed by an example translation given in Fig. 1.

First we define some additional notation. We assume a unique numbering of the events (and conditions) of the finite complete prefix. We use the notation e_i (b_i) to refer to the event (condition) number i. In the logic programs $\mathtt{e_i}$, ($\mathtt{b_i}$) is an atom of the logic program corresponding to the event e_i (condition b_i). In the logic program definitions we use the convention that a part of a rule will be omitted, if the corresponding set evaluates to the empty set. For example rule of type 1 of Def. 4.1 below for an event e_i, such that ${}^\bullet({}^\bullet e_i) = \emptyset$, would become: $\mathtt{e_i} \leftarrow \mathit{not}\,(\mathtt{be_i})$.

Definition 4.1. Let $\beta = \langle N, h\rangle$ with $N = \langle B, E, F\rangle$ be a finite complete prefix of a given 1-safe net system Σ. Let $P_B(\beta)$ be a logic program containing the following rules:

1. For all $e_i \in E \setminus \mathit{CutOffs}(\beta)$ a rule:
   $\mathtt{e_i} \leftarrow \mathtt{e_{p_1}}, \ldots, \mathtt{e_{p_n}}, \mathit{not}\,(\mathtt{be_i})$,
   such that $\{e_{p_1}, \ldots, e_{p_n}\} = {}^\bullet({}^\bullet e_i)$.

2. For all $e_i \in E \setminus \mathit{CutOffs}(\beta)$ a rule:
   $\mathtt{be_i} \leftarrow \mathit{not}\,(\mathtt{e_i})$.

3. For all $b_i \in B$ such that $|b_i{}^\bullet \setminus \mathit{CutOffs}(\beta)| \geq 2$ a rule:
   $\leftarrow 2\{\mathtt{e_{p_1}}, \ldots, \mathtt{e_{p_n}}\}$,
   such that $\{e_{p_1}, \ldots, e_{p_n}\} = b_i{}^\bullet \setminus \mathit{CutOffs}(\beta)$.

The intuition behind the rules of the program $P_B(\beta)$ are the following. Rules of the type 1 give the preconditions under which an atom corresponding to each event can exist in a configuration. Rules of the type 2 enable an event not to be included in a configuration even if its preconditions are present. Rules of the type 3 disallow all sets of events which contain events in a conflict. Note that because in prefixes each condition has only one event in its preset, the program above does

not need atoms corresponding to the conditions of the prefix. The translation above could be trivially extended to also include the cut-off events, but they are not needed by the applications in this work.

We define a mapping from a set of events of the prefix to a set of atoms of a logic program and vice versa.

Definition 4.2. The set of atoms of a logic program P corresponding to a set of events $C \subseteq E \setminus \mathit{Cutoffs}(\beta)$ of a finite complete prefix β is $\mathit{Model}(C) = \{\mathtt{e_i} \mid e_i \in C\} \cup \{\mathtt{be_j} \mid e_j \in E \setminus \{C \cup \mathit{Cutoffs}(\beta)\}\}$.

Definition 4.3. The set of events corresponding to a stable model Δ of a logic program P is $\mathit{Events}(\Delta) = \{e_i \in E \mid \mathtt{e_i} \in \Delta\}$.

Now we are ready to state the correspondence between the finite complete prefix and the basic part of our translation.

Theorem 4.1. *Let β be a finite complete prefix of a 1-safe net system Σ, let $P_B(\beta)$ be the logic program translation by Def. 4.1, and let C be a configuration of β, such that $C \cap \mathit{Cutoffs}(\beta) = \emptyset$. Then the set of atoms $\Delta = \mathit{Model}(C)$ is a stable model of $P_B(\beta)$. Additionally, the mapping $\mathit{Events}(\Delta)$ is a bijective mapping from the stable models of $P_B(\beta)$ to the configurations of β which contain no cut-off events.*

Proof:
See Appendix A. □

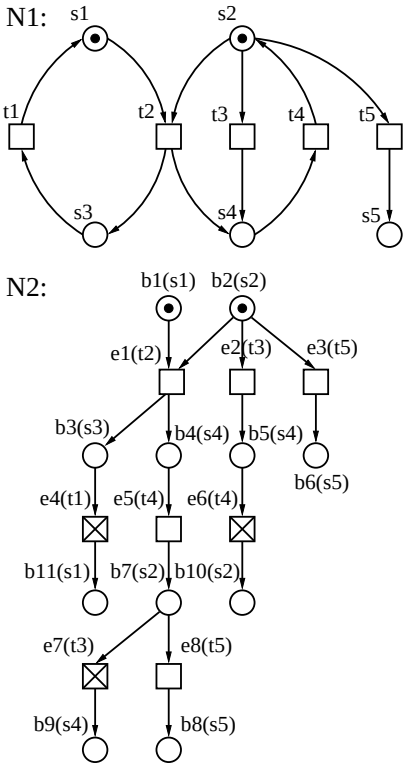
Next we move to the deadlock translation. We add a set of rules to the program which place additional constraints on the stable models of the program $P_B(\beta)$. We add integrity rules to the program, which remove all stable models of the basic program which are not deadlocks. To do this we model the the enabling of each event (cut-off or not) of the prefix in the logic program.

Definition 4.4. Let β be a finite complete prefix of a given 1-safe net system Σ. Let $P_D(\beta)$ be a logic program containing all the rules of the program $P_B(\beta)$ of Def. 4.1, and also the following rules:

1. For all $b_i \in \{b_j \in B \mid b_j{}^\bullet \neq \emptyset\}$ a rule:
   $\mathtt{b_i} \leftarrow \mathtt{e_1}$, *not* $(\mathtt{e_{p_1}})$, ... , *not* $(\mathtt{e_{p_n}})$,
   such that $\{e_l\} = {}^\bullet b_i$, and $\{e_{p_1}, \ldots, e_{p_n}\} = b_i{}^\bullet \setminus \mathit{CutOffs}(\beta)$.

2. For all $e_i \in E$ a rule:
   $\leftarrow \mathtt{b_{p_1}}, \ldots, \mathtt{b_{p_n}}$,
   such that $\{b_{p_1}, \ldots, b_{p_n}\} = {}^\bullet e_i$.

Theorem 4.2. *Let β be a finite complete prefix of a 1-safe net system Σ, and let $P_D(\beta)$ be the logic program translation by Def. 4.4. There exists a stable model of $P_D(\beta)$ iff Σ has a reachable deadlock marking. Additionally, for any stable model Δ of $P_D(\beta)$, the set of events $C = \mathit{Events}(\Delta)$ is a deadlock configuration of β, such that $\mathit{Mark}(C)$ is a reachable deadlock marking of Σ.*



$P_D(N2)$:

$\mathtt{e}_1 \leftarrow \mathit{not}\,(\mathtt{be}_1)$
$\mathtt{be}_1 \leftarrow \mathit{not}\,(\mathtt{e}_1)$
$\mathtt{e}_2 \leftarrow \mathit{not}\,(\mathtt{be}_2)$
$\mathtt{be}_2 \leftarrow \mathit{not}\,(\mathtt{e}_2)$
$\mathtt{e}_3 \leftarrow \mathit{not}\,(\mathtt{be}_3)$
$\mathtt{be}_3 \leftarrow \mathit{not}\,(\mathtt{e}_3)$
$\mathtt{e}_5 \leftarrow \mathtt{e}_1, \mathit{not}\,(\mathtt{be}_5)$
$\mathtt{be}_5 \leftarrow \mathit{not}\,(\mathtt{e}_5)$
$\mathtt{e}_8 \leftarrow \mathtt{e}_5, \mathit{not}\,(\mathtt{be}_8)$
$\mathtt{be}_8 \leftarrow \mathit{not}\,(\mathtt{e}_8)$
$\leftarrow 2\{\mathtt{e}_1, \mathtt{e}_2, \mathtt{e}_3\}$
$\mathtt{b}_1 \leftarrow \mathit{not}\,(\mathtt{e}_1)$
$\mathtt{b}_2 \leftarrow \mathit{not}\,(\mathtt{e}_1), \mathit{not}\,(\mathtt{e}_2), \mathit{not}\,(\mathtt{e}_3)$
$\mathtt{b}_3 \leftarrow \mathtt{e}_1$
$\mathtt{b}_4 \leftarrow \mathtt{e}_1, \mathit{not}\,(\mathtt{e}_5)$
$\mathtt{b}_5 \leftarrow \mathtt{e}_2$
$\mathtt{b}_7 \leftarrow \mathtt{e}_5, \mathit{not}\,(\mathtt{e}_8)$
$\leftarrow \mathtt{b}_1, \mathtt{b}_2$
$\leftarrow \mathtt{b}_2$
$\leftarrow \mathtt{b}_3$
$\leftarrow \mathtt{b}_4$
$\leftarrow \mathtt{b}_5$
$\leftarrow \mathtt{b}_7$

Figure 1. Deadlock translation example.

Proof:
See Appendix A. □

In Fig. 1 an example of the deadlock translation is given. The prefix $N2$ is a finite complete prefix of the 1-safe nets system $N1$. The cut-off events of $N2$ are marked with crosses. The translated program $P_D(N2)$ has only one stable model $\Delta = \{\mathtt{be}_1, \mathtt{be}_2, \mathtt{e}_3, \mathtt{be}_5, \mathtt{be}_8, \mathtt{b}_1\}$, and the set $\mathit{Events}(\Delta) = \{e_3\}$ is a deadlock configuration of $N2$.

Next we will present a way of translating reachability problems. First we need a way of making statements about an individual marking.

Definition 4.5. An *assertion* on a marking of a 1-safe net system $\Sigma = \langle S, T, F, M_0 \rangle$ is a tuple $\langle S^+, S^- \rangle$, where $S^+, S^- \subseteq S$, and $S^+ \cap S^- = \emptyset$. The assertion $\langle S^+, S^- \rangle$ *agrees* with a marking M of Σ iff:

$$S^+ \subseteq \{s \in S \,|\, M(s) = 1\} \wedge S^- \subseteq \{s \in S \,|\, M(s) = 0\}.$$

With assertions we can easily formulate both the reachability and submarking reachability problems. The idea is again to add some integrity rules to the program which remove all stable

models of $P_B(\beta)$ which do not agree with the assertion. The basic structure is the same as for deadlocks, only a set of atoms which represent the marking of the original net are added.

Definition 4.6. Let β be a finite complete prefix of a given 1-safe net system $\Sigma = \langle S, T, F, M_0 \rangle$, and let $\phi = \langle S^+, S^- \rangle$ be an assertion on the places of Σ. Let $P_R(\beta, \phi)$ be a logic program containing all the rules of the program $P_B(\beta)$ of Def. 4.1, and also the following rules:

1. For all $b_i \in \{b_j \in B \mid h(b_j) \in S^+ \cup S^- \wedge {}^\bullet b_j \in E \setminus \mathit{Cutoffs}(\beta)\}$ a rule:
   $\mathtt{b_i} \leftarrow \mathtt{e_1}$, *not* $(\mathtt{e_{p_1}})$, $\ldots$, *not* $(\mathtt{e_{p_n}})$,
   such that $\{e_l\} = {}^\bullet b_i$, and $\{e_{p_1}, \ldots, e_{p_n}\} = b_i{}^\bullet \setminus \mathit{CutOffs}(\beta)$.

2. For all $b_i \in \{b_j \in B \mid h(b_j) \in S^+ \cup S^- \wedge {}^\bullet b_j \in E \setminus \mathit{Cutoffs}(\beta)\}$ a rule:
   $\mathtt{s_i} \leftarrow \mathtt{b_i}$,
   such that $s_i = h(b_i)$.

3. For all $s_i \in S^+$ a rule:
   $\leftarrow$ *not* $(\mathtt{s_i})$.

4. For all $s_i \in S^-$ a rule:
   $\leftarrow \mathtt{s_i}$.

Note that only conditions of the prefix β and places of Σ which can affect the assertion ϕ are translated. Also cut-off postset conditions are not translated, because cut-offs will not be fired.

Theorem 4.3. *Let β be a finite complete prefix of a 1-safe net system Σ, and let $P_R(\beta, \phi)$ be a logic program translation by Def. 4.6. The logic program $P_R(\beta, \phi)$ has a stable model iff there exists a reachable marking of Σ which agrees with ϕ. Additionally, for any stable model Δ of $P_R(\beta, \phi)$, the configuration $C = \mathit{Events}(\Delta)$ is a configuration of β, such that $\mathit{Mark}(C)$ is a reachable marking of Σ which agrees with ϕ.*

Proof:
See Appendix A. □

It is easy to see that the sizes of all the translations presented are linear in the size of the prefix β, i.e. $\mathcal{O}(|B| + |E| + |F|)$. Because the rule-based constraint programming system we use needs linear space in the size of the input program, deadlock and reachability property checking exploiting these translations can be made using linear space in the size of the prefix. The translations are also local, which makes them straightforward to implement using linear time in the size of the prefix.

5. Deadlock Property Checking Implementation

We have implemented the deadlock property checking translation, and we plan on implementing the reachability translation in the near future. The translation reads a file containing the

description of a finite complete prefix generated by the PEP-tool [8]. It generates a logic program using the deadlock translation, which is then through an internal interface given to the `smodels` stable model generator. The translation performs the following optimizations:

1. Not generating the program iff the number of cut-off events is zero.
2. Removal of blocking of "stubborn" transitions: If we find an event e_i such that $({}^\bullet e_i)^\bullet \setminus \mathit{Cutoffs}(\beta) = \{e_i\}$, the corresponding rule of type 1 of the program $P_B(\beta)$ is replaced by a rule of the form: $\mathtt{e_i} \leftarrow \mathtt{e_{p_1}}, \ldots, \mathtt{e_{p_n}}$, and the rule 2 of the form: $\mathtt{be_i} \leftarrow \mathit{not}\,(\mathtt{e_i})$ is not created. Also the corresponding liveness rule of type 2 of the program $P_D(\beta)$ of the form: $\leftarrow \mathtt{b_{p_1}}, \ldots, \mathtt{b_{p_n}}$ does not need to be created as far as the event e_i is concerned.
3. Removal of redundant condition rules: The rule of type 1 of the program $P_D(\beta)$ corresponding to condition b_i is removed if the atom $\mathtt{b_i}$ is does not appear elsewhere in the program.
4. Removal of redundant atoms: If a rule of the form: $\mathtt{a_1} \leftarrow \mathtt{a_2}$ would be generated, and this is the only rule in which $\mathtt{a_1}$ appears as a head, then all instances of $\mathtt{a_1}$ are replaced by $\mathtt{a_2}$, and the rule is discarded.
5. Duplicate rule removal: Only one copy of each rule is generated.

For the optimization 1 it is easy to see that the net system Σ will deadlock, because the finite complete prefix is finite and does not contain any cut-offs. Thus the net system Σ can fire only a finite number of transitions. It also is straightforward to prove that the optimizations 3-5 do not alter the number of stable models the program has. The optimization 2 is motivated by stubborn sets [19]. The intuition is that whenever e_i is enabled, it must be disabled in order to reach a deadlock. However the only way of disabling e_i is to fire it. Therefore we can discard all configurations in which e_i is enabled as not being deadlock configurations.

We argue that optimization 2 is correct, i.e. the stable models of the program $P_D(\beta)$ are not affected by it (modulo the possible removal of the atom $\mathtt{be_i}$ from the set of atoms of the optimized program). Consider the original program, and an optimized one in which an event e_i has been optimized using optimization 2. If we look only at the two programs without the deadlock detection parts added by Def. 4.4, their only difference is that in the original program it is possible to leave the event e_i enabled but not fired, while this is not possible in the optimized program. Thus clearly the set of stable models of the optimized program is a subset of the stable models of the original one. If we have any configuration in which the event e_i is enabled but is not fired, then the set of atoms corresponding to this configuration is not a stable model of the original program. This is the case because the integrity rule of type 2 of Def. 4.4 corresponding to the event e_i eliminates such a potential stable model. Therefore the optimized program will have the same number of stable models as the original one.

We do quite an extensive set of optimizations. The optimizations 1 and 2 are deadlock detection specific. The optimizations 3-5 can be seen as general logic program optimizations based on static analysis, and could in principle be done in the stable model generator after the translation. The optimizations 1-4 are implemented using linear time and space in the size of the prefix. The duplicate rule removal is implemented with hashing.

We use succinct rule encodings with extended rules when possible. The two rules $\mathtt{e_i} \leftarrow \mathtt{e_{p_1}}, \ldots, \mathtt{e_{p_n}}$, *not* $(\mathtt{be_i})$, and $\mathtt{be_i} \leftarrow$ *not* $(\mathtt{e_i})$ can be more succinctly encoded by an extended rule of the form: $\{\mathtt{e_i}\} \leftarrow \mathtt{e_{p_1}}, \ldots, \mathtt{e_{p_n}}$. Also $\leftarrow 2\{\mathtt{a_1}, \mathtt{a_2}\}$ is replaced by: $\leftarrow \mathtt{a_1}, \mathtt{a_2}$. We also sort the rules after the translation. In our experiments the sorting seems to have only a minimal effect on the total running time, but produces nicer looking logic program (debugging) output.

After the translation has been created, the `smodels` computational engine is used to check whether a stable model of the program exists. If one exists, the deadlock checker outputs an example deadlock configuration using the found stable model. Otherwise the program tells that the net is deadlock free.

5.1. Experimental Results

We have made experiments with our approach using examples by Corbett [2], McMillan [12, 13], and Melzer and Römer [14]. They were previously used by Melzer and Römer in [14] and by Best and Römer in [1], where additional information about them can be found. We compare our approach with two other finite complete prefix based deadlock checking methods. The first method is the branch-and-bound deadlock detection algorithm by McMillan [12, 13, 14], and the other is the mixed integer programming approach by Melzer and Römer [14].

The Figures 2-4 present the running times in seconds for the various algorithms used in this work, and for the mixed integer programming approach those presented in [14]. The running times have been measured using a Pentium 166MHz, 64MB RAM, 128MB swap, Linux 2.0.29, g++ 2.7.2.1, `smodels` pre-2.0.30, McMillan's algorithm version 2.1.0 by Stefan Römer, and PEP 1.6g. The experiments with the mixed integer programming approach by Melzer and Römer used a commercial MIP-solver CPLEX, and were conducted on a Sparcstation 20/712, 96MB RAM.

The rows of the tables correspond to different problems. The columns represent: sum of user and system times measured by `/usr/bin/time` command, or times reported in [14], depending on the column:

- Unf = time for unfolding (creation of the finite complete prefix) (PEP).
- $\mathrm{DC_{MIP}}$ = time for Mixed integer programming approach in [14].
- $\mathrm{DC_{McM}}$ = time for McMillan's algorithm, average of 4 runs.
- $\mathrm{DC_{smo}}$ = time for `smodels` based deadlock checker, average of 4 runs.

The marking $vm(n)$ notes that the program ran out of virtual memory after n seconds. The other fields of the figures are as follows: $|B|$: number of conditions, $|E|$: number of events, #c: number of cut-off events, DL: Y - the net system has a deadlock, CP: *choice points* i.e. the number of nondeterministic guesses `smodels` did during the run. The $\mathrm{DC_{smo}}$ column also includes the logic program translation time, which was always under 10 seconds for the examples.

The logic programming approach using the `smodels` system was able to produce an answer for all the examples presented here, while the McMillan's algorithm implementation ran out of virtual memory on some of the larger examples. Our approach was sometimes much faster, see

Problem(size)	$\|B\|$	$\|E\|$	#c	DL	CP	Unf[1]	DC^{2}_{MIP}	DC^{1}_{McM}	DC^{1}_{smo}
DPD(5)	1582	790	211	N	0	0.6	17.3	1.6	1.0
DPD(6)	3786	1892	499	N	0	3.2	82.8	12.3	6.1
DPD(7)	8630	4314	1129	N	0	17.4	652.6	128.9	31.4
DPH(5)	2712	1351	547	N	0	1.3	42.9	6.5	1.8
DPH(6)	14474	7231	3377	N	0	33.7	1472.8	1063.7	32.9
DPH(7)	81358	40672	21427	N	0	929.3	-	*vm*(1690.2)	760.6
ELEVATOR(2)	1562	827	331	Y	2	0.6	2.3	0.5	0.7
ELEVATOR(3)	7398	3895	1629	Y	3	10.3	14.5	10.1	15.0
ELEVATOR(4)	32354	16935	7337	Y	4	186.1	387.8	268.8	231.7
FURNACE(1)	535	326	189	N	0	0.1	0.3	0.2	0.0
FURNACE(2)	5139	3111	1990	N	0	3.2	18.1	11.1	0.6
FURNACE(3)	34505	20770	13837	N	0	134.7	1112.5	*vm*(392.5)	7.1
RING(7)	813	403	79	N	0	0.2	17.1	0.2	0.4
RING(9)	1599	795	137	N	0	0.7	71.2	0.7	2.2
RW(9)	9272	4627	4106	N	0	2.0	58.5	68.2	0.4
RW(12)	98378	49177	45069	N	0	137.5	24599.9	*vm*(3050.5)	4.2

Figure 2 Measured running times in seconds:
[1] = Pentium 166MHz, 64MB RAM, Linux 2.0.29.
[2] = Sparcstation 20/712, 96MB RAM [14].

e.g. FURNACE(3), RW(12), SYNC(3), BDS(1), GASQ(4), and Q(1). The McMillan's algorithm was faster than our approach on the following problem classes: RING, HART, SENT and SPD. These problems are quite easy for both methods, running times for the first three were a few seconds, and for the fourth still well under 30 seconds. On the DME and KEY examples our approach is scaling better as the problem sizes increase. McMillan's algorithm is most competitive when the number of cut-off events is relatively small.

We do not have access to the MIP-solver used in [14], and also our experiments in [9] seem to indicate that the computer we made our experiments on is faster than theirs. This makes it difficult to comment on the absolute running times between different machines. However our approach is scaling better on most examples, see e.g. RW, DME, and SYNC examples.

An observation that should be made is that the number of choice points for `smodels` in these examples is very low, with a maximum of 9 choice points in the example SPD(1). This means that on this example set the search space pruning techniques were very effective in minimizing the number of nondeterministic choices that were needed to solve the examples.

The example nets and C++ source code for our translation including `smodels` are available from the author.

Problem(size)	$\|B\|$	$\|E\|$	#c	DL	CP	Unf[1]	DC^{2}_{MIP}	DC^{1}_{McM}	DC^{1}_{smo}
DME(4)	2381	652	16	N	0	1.1	216.1	1.4	3.9
DME(5)	4096	1145	25	N	0	3.2	1968.3	5.5	13.7
DME(6)	6451	1830	36	N	0	8.5	13678.3	20.1	38.0
DME(7)	9542	2737	49	N	0	18.1	-	66.1	86.7
DME(8)	13465	3896	64	N	0	37.0	-	196.0	182.3
DME(9)	18316	5337	81	N	0	70.0	-	542.2	366.6
DME(10)	24191	7090	100	N	0	124.0	-	1268.4	646.1
DME(11)	31186	9185	121	N	0	207.0	-	3070.9	1134.8
SYNC(2)	4007	2162	490	N	0	4.6	171.6	37.0	1.8
SYNC(3)	29132	15974	5381	N	0	218.6	11985.0	14073.3	66.5

Figure 3 Measured running times in seconds:
[1] = Pentium 166MHz, 64MB RAM, Linux 2.0.29.
[2] = Sparcstation 20/712, 96MB RAM [14].

6. Conclusions

Our main contribution is a method to transform the deadlock and reachability problems for 1-safe Petri nets into the problem of finding a stable model of a logic program and its correctness proof. We do the translation in two steps: (i) Existing methods and tools are used to generate a finite complete prefix of the 1-safe Petri net [5, 8, 12, 13]. (ii) The deadlock and reachability problems for the finite complete prefix are translated into the problem of finding a stable model of a logic program. This step uses the two new translations presented in this work, both of which are linear in the size of the prefix.

Correctness proofs of are done in two steps. First a program is constructed whose stable models are proved to have a one-to-one correspondence with the configurations of the finite complete prefix which contain no cut-off events. Then additional rules added by the translations are shown to either remove all potential stable models corresponding to live configurations, or all potential stable models which do not agree with the used assertion, depending on the translation.

We present experimental results to support the feasibility of this approach for the deadlock detection problem. We use an existing constraint-based logic programming framework, the `smodels` system, for solving the problem of finding a stable model of a logic program. Our experiments show that the approach seems to be quite robust and competitive on the examples available to us. More experiments are needed to evaluate the feasibility of the approach on the reachability problem.

There are interesting topics for future research. It seems possible to extend the translations to allow for a larger class of Petri nets to be translated, while still keeping the problem NP-complete. McMillan's algorithm can be seen to be more goal directed algorithm than our approach, and an alternative translation using the basic ideas of McMillan's algorithm could be created. The `smodels` system is quite a general purpose constraint propagation based search engine. Creating specialized algorithms for the two problems at hand could further improve the

Problem(size)	$\|B\|$	$\|E\|$	#c	DL	CP	Unf[1]	DC^1_{McM}	DC^1_{smo}
BDS(1)	12310	6330	3701	N	0	18.3	171.9	4.1
FTP(1)	178077	89042	35247	N	0	6470.5	*vm*(5413.1)	2080.0
GASN(3)	2409	1205	401	N	0	1.2	13.2	2.4
GASN(4)	15928	7965	2876	N	0	49.3	2630.4	105.5
GASN(5)	100527	50265	18751	N	0	1972.7	*vm*(3393.7)	3958.4
GASQ(3)	2593	1297	490	N	0	1.3	10.1	2.4
GASQ(4)	19864	9933	4060	N	0	72.9	4170.3	127.5
OVER(4)	1561	797	240	N	0	0.6	0.9	0.1
OVER(5)	7388	3761	1251	N	0	11.9	38.1	0.9
HART(50)	354	202	1	Y	5	0.1	0.0	0.2
HART(75)	529	302	1	Y	6	0.3	0.1	0.4
HART(100)	704	402	1	Y	6	0.4	0.1	0.8
KEY(2)	1304	650	201	Y	5	0.5	0.3	0.7
KEY(3)	13885	6940	2921	Y	5	41.0	38.8	68.4
KEY(4)	135556	67775	32081	Y	8	3457.8	*vm*(3930.9)	4418.7
MMGT(3)	11575	5841	2529	Y	0	22.6	592.4	20.0
MMGT(4)	92940	46902	20957	Y	0	1466.2	*vm*(3068.0)	1375.2
Q(1)	16090	8402	1173	Y	5	89.5	71.2	4.7
SENT(75)	533	266	40	Y	6	0.2	0.1	0.3
SENT(100)	608	291	40	Y	6	0.3	0.1	0.4
SPD(1)	5317	3138	1311	Y	9	6.1	8.4	21.8

Figure 4 Measured running times in seconds:
[1] = Pentium 166MHz, 64MB RAM, Linux 2.0.29.

competitiveness of our approach. The subject of applying our approach to some form of model checking is a very interesting area for future research.

7. Acknowledgements

The author would like to thank Ilkka Niemelä for introducing him into the rule-based constraint programming framework, and for many constructive ideas for this paper. The tool `smodels` was programmed by Patrik Simons, who gave valuable support for its usage. Stephan Melzer and Stefan Römer provided the example nets, and also Linux binaries for McMillan's algorithm, which both were invaluable. Thanks to Burkhard Graves and Bernd Grahlmann for supplying C source code to read PEP prefix files. The financial support of Helsinki Graduate School on Computer Science and Engineering (HeCSE), and the Academy of Finland are gratefully acknowledged.

References

[1] E. Best. Partial order verification with PEP. In G. Holzmann, D. Peled, and V. Pratt, editors, *Proceedings of POMIV'96, Workshop on Partial Order Methods in Verification.*

American Mathematical Society, July 1996.

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

[3] J. Engelfriet. Branching processes of Petri nets. In *Acta Informatica 28*, pages 575–591, 1991.

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

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

[6] M. Fitting. *First-Order Logic and Automated Theorem Proving.* Springer-Verlag, New York, 1990.

[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] B. Grahlmann. The PEP Tool. In *Proceeding of 9th International Conference on Computer Aided Verification (CAV'97)*, pages 440–443. Springer-Verlag, June 1997. LNCS 1254.

[9] K. Heljanko. Deadlock checking for complete finite prefixes using logic programs with stable model semantics (extended abstract). In *Proceedings of the Workshop Concurrency, Specification & Programming 1998*, Informatik-Bericht Nr. 110, pages 106–115. Humboldt-University, Berlin, September 1998.

[10] K. Heljanko. Using logic programs with stable model semantics to solve deadlock and reachability problems for 1-safe Petri nets. In *Proceedings of Fifth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99)*, pages 240–254. Springer-Verlag, Berlin, March 1999. LNCS 1579.

[11] W. Marek and M. Truszczyński. Autoepistemic logic. *Journal of the ACM*, 38:588–619, 1991.

[12] K. L. McMillan. Using unfoldings to avoid the state space explosion problem in the verification of asynchronous circuits. In *Proceeding of 4th Workshop on Computer Aided Verification (CAV'92)*, pages 164–174, 1992. LNCS 663.

[13] K. L. McMillan. A technique of a state space search based on unfolding. In *Formal Methods is System Design 6(1)*, pages 45–65, 1995.

[14] 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. LNCS 1254.

[15] I. Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. In *Proceedings of the Workshop on Computational Aspects of Nonmonotonic Reasoning*, pages 72–79, Trento, Italy, May 1998. Helsinki University of Technology, Digital Systems Laboratory, Research Report A52.

[16] I. Niemelä and P. Simons. Smodels – an implementation of the stable model and well-founded semantics for normal logic programs. In *Proceedings of the 4th International Conference on Logic Programming and Non-Monotonic Reasoning*, pages 420–429, Dagstuhl, Germany, July 1997. Springer-Verlag.

[17] P. Simons. Towards constraint satisfaction through logic programs and the stable model semantics. Research Report A47, Helsinki University of Technology, Espoo, Finland, August 1997. Licenciate's thesis.

[18] P. Simons. Extending the stable model semantics with more expressive rules, 1998. Unpublished manuscript.

[19] A. Valmari. A stubborn attack on state explosion. *Formal Methods in System Design*, 1 (1992):297–322.

[20] A. Van Gelder, K.A. Ross, and J.S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, July 1991.

Appendix A

Proof of Theorem 4.1

We prove the correspondence between the stable models of program $P_B(\beta)$ and the configurations of β which contain no cut-off events. We proceed by first proving auxiliary Lemma 1. In Lemmas 2 and 3 one direction of the correspondence between stable models of the program $P_B(\beta)$ and the configurations of β with no cut-off events is proved. The Lemma 4 shows the other direction of the correspondence, and together with Corollary 1 shows that the mapping $Events(\Delta)$ is a bijective mapping between the two sets in question.

Lemma 1. *Let Δ be a stable model of program $P_B(\beta)$ obtained from the translation by Def. 4.1, and $\mathtt{e_i}, \mathtt{be_i} \in Atoms(P_B(\beta))$. Now the following holds: $\mathtt{e_i} \in \Delta \Rightarrow \mathtt{be_i} \in \overline{\Delta}$, and $\mathtt{e_i} \in \overline{\Delta} \Rightarrow \mathtt{be_i} \in \Delta$.*

Proof:
Assume that $\mathtt{e_i} \in \Delta$. The only rule in which $\mathtt{be_i}$ appears as a head is a rule of type 2 of the form $\mathtt{be_i} \leftarrow not\,(\mathtt{e_i})$. By the definition of stable models, because $\mathtt{e_i} \in \Delta$, the reduct $R\,(P_B(\beta), not(\overline{\Delta}))$ does not contain this rule, and thus $\mathtt{be_i} \notin Dcl(P_B(\beta), not(\overline{\Delta}))$, which implies $\mathtt{be_i} \in \overline{\Delta}$, because Δ is a stable model.

Assume that $\mathtt{e_i} \in \overline{\Delta}$. Therefore $R\,(P_B(\beta), not(\overline{\Delta}))$ contains the rule $\mathtt{be_i} \leftarrow$, and thus $\mathtt{be_i} \in Dcl(P_B(\beta), not(\overline{\Delta}))$, which implies $\mathtt{be_i} \in \Delta$, because Δ is a stable model. □

Corollary 1. *Let Δ be any stable model of $P_B(\beta)$. The set of events $C = Events(\Delta)$* fully specifies *Δ, i.e. the following holds: $Model(Events(\Delta)) = \Delta$.*

Now we do the main proof of Theorem 4.1. As the first step we use a subset of the rules of the program $P_B(\beta)$ and prove that for this subset of rules the set of atoms $\Delta = Model(C)$ is a stable model.

Lemma 2. *Let C be a configuration of β such that $C \cap Cutoffs(\beta) = \emptyset$, and let $P_A(\beta)$ be the logic program containing only the rules of the types 1 and 2 of the program $P_B(\beta)$. The set of atoms $\Delta = Model(C)$ is a stable model of $P_A(\beta)$.*

Proof:
We need to prove that $Dcl(P_A(\beta), not(\overline{\Delta})) = \Delta$. We proceed by case analysis on the pairs of atoms $\{\mathtt{e_i}, \mathtt{be_i}\}$ in $Atoms(P_A(\beta))$ showing for both atoms that: For each atom $a \in \Delta$ that $a \in Dcl(P_A(\beta), not(\overline{\Delta}))$, and for each atom $a \in \overline{\Delta}$ that $a \notin Dcl(P_A(\beta), not(\overline{\Delta}))$.

1. Investigate the pair of atoms $\{\mathtt{e_i}, \mathtt{be_i}\}$ such that $\mathtt{e_i} \in \overline{\Delta}$, and thus by Def. 4.2: $\mathtt{be_i} \in \Delta$. The reduct $R\,(P_A(\beta), not(\overline{\Delta}))$ contains a rule of the form $\mathtt{be_i} \leftarrow$, and also the only way of deducing $\mathtt{e_i}$, the rule of type 1 of the form: $\mathtt{e_i} \leftarrow \mathtt{e_{p_1}}, \ldots, \mathtt{e_{p_n}}$, is not in the reduct, which implies $\mathtt{be_i} \in Dcl(P_A(\beta), not(\overline{\Delta}))$ and $\mathtt{e_i} \notin Dcl(P_A(\beta), not(\overline{\Delta}))$.
2. Investigate the pair of atoms $\{\mathtt{e_i}, \mathtt{be_i}\}$ such that $\mathtt{e_i} \in \Delta$, and thus by Def. 4.2: $\mathtt{be_i} \in \overline{\Delta}$. The reduct $R\,(P_A(\beta), not(\overline{\Delta}))$ does not contain the rule of type 2 of the form: $\mathtt{be_i} \leftarrow$, which implies $\mathtt{be_i} \notin Dcl(P_A(\beta), not(\overline{\Delta}))$. The reduct contains a rule of type 1 of the form: $\mathtt{e_i} \leftarrow \mathtt{e_{p_1}}, \ldots, \mathtt{e_{p_n}}$, such that $\{e_{p_1}, \ldots, e_{p_n}\} = {}^\bullet({}^\bullet e_i)$. We need an induction to complete the proof for this case.

What is left to be proved is that $\mathtt{e_j} \in Dcl(P_A(\beta), not(\overline{\Delta}))$ for all $\mathtt{e_j} \in \Delta$, and thus for all $e_j \in C$. If $C = \emptyset$ we are done. Otherwise, we prove the previous claim by induction on the index number k of a sequence of events: $e'_1, \ldots, e'_{|C|}$, such that $\{e'_1, \ldots, e'_{|C|}\} = C$, and for all $k \in \{1, \ldots, |C|\}$ it holds that ${}^\bullet({}^\bullet e'_k) \subseteq \bigcup_{1 \le l < k} \{e'_l\}$. Because C is a configuration, such a sequence (a causal total order of events of a configuration) must exist. We pick one such a sequence.

- *Base case $k = 1$:* For e'_1 it holds that ${}^\bullet({}^\bullet e'_1) = \emptyset$. Thus the reduct has a rule of the form $\mathtt{e'_1} \leftarrow$, which implies $\mathtt{e'_1} \in Dcl(P_A(\beta), not(\overline{\Delta}))$.
- *Inductive case $k > 1$:* For all $1 \le j < k$ the claim $\mathtt{e'_j} \in Dcl(P_A(\beta), not(\overline{\Delta}))$ holds by the inductive hypothesis. Because ${}^\bullet({}^\bullet e'_k) \subseteq \bigcup_{1 \le l < k} \{e'_l\}$, the reduct has a rule of the form $\mathtt{e'_k} \leftarrow \mathtt{e_{p_1}}, \ldots, \mathtt{e_{p_n}}$, such that $\{\mathtt{e_{p_1}}, \ldots, \mathtt{e_{p_n}}\} \subseteq Dcl(P_A(\beta), not(\overline{\Delta}))$ by the inductive hypothesis, which implies $\mathtt{e'_k} \in Dcl(P_A(\beta), not(\overline{\Delta}))$.

The union of all the pairs equals $Atoms(P_A(\beta))$, which implies Δ is a stable model of $P_A(\beta)$. □

Now we continue our proof by considering the full program $P_B(\beta)$ in Lemma 3.

Lemma 3. *Let C be a configuration of β such that $C \cap Cutoffs(\beta) = \emptyset$. The set of atoms $\Delta = Model(C)$ is a stable model of the program $P_B(\beta)$.*

Proof:
By Lemma 2: $\Delta = Model(C)$ is a stable model of $P_A(\beta)$. In the program $P_B(\beta)$ only integrity rules of type 3 of Def. 4.1 have been added. Thus the set of stable models of $P_B(\beta)$ is always a subset of the stable models of $P_A(\beta)$. Because C is a configuration, there does not exists two non-cut-off events e_i and e_j, and a condition b_k such that: $e_i \neq e_j$, and $b_k \in {}^\bullet e_i \cap {}^\bullet e_j$. Therefore there is no integrity rule which could be used, which implies Δ is a stable model of $P_B(\beta)$. □

Lemma 4. *If Δ is a stable model of the program $P_B(\beta)$, then the set of events $C = Events(\Delta)$ is a configuration of β such that $C \cap Cutoffs(\beta) = \emptyset$.*

Proof:
Because Corollary 1 says that any stable model is fully specified by a set of events, to get a contradiction we need to find a set of events E', which is not a configuration, $E' \cap Cutoffs(\beta) = \emptyset$, and $\Delta = Model(E')$ is a stable model of $P_B(\beta)$. There are two cases:

1. Assume that E' is not causally closed. Thus there must exist an event $e_i \in E'$, such that ${}^\bullet({}^\bullet e_i) \setminus E' \neq \emptyset$. Now by Def. 4.2 and Lemma 1: $\mathtt{be_i} \notin Model(E')$, and thus the reduct $R\,(P_B(\beta), not(\overline{\Delta}))$ contains a rule of the form $\mathtt{e_i} \leftarrow \mathtt{e_{p_1}}, \ldots, \mathtt{e_{p_n}}$, which is the only rule in which $\mathtt{e_i}$ appears as a head. However by ${}^\bullet({}^\bullet e_i) \setminus E' \neq \emptyset$ and Def. 4.2: $\mathtt{be_j} \in Model(E')$ for some $\mathtt{j} \in \{\mathtt{p_1}, \ldots, \mathtt{p_n}\}$. Thus by Lemma 1: $\mathtt{e_j} \notin \Delta$, and thus $\mathtt{e_i} \notin Dcl(P_B(\beta), not(\overline{\Delta}))$, which implies Δ is not a stable model.
2. Assume that E' contains a conflict. Thus there exists two non-cut-off events e_i and e_j, and a condition b_k such that: $e_i \neq e_j$, and $b_k \in {}^\bullet e_i \cap {}^\bullet e_j$. The integrity rule of type 3 of Def. 4.1 corresponding to the condition b_k has the form $\leftarrow 2\{\mathtt{e_{p_1}}, \ldots, \mathtt{e_{p_n}}\}$, such that $\{\mathtt{e_i}, \mathtt{e_j}\} \subseteq \Delta$, which eliminates the possibility that Δ is a stable model.

□

The Lemmas 3 and 4 prove the correspondence between the configurations of the prefix β which contain no cut-off events, and the stable models of the program $P_B(\beta)$. Combined with Corollary 1 the bijectivity between these two sets is shown. This completes the proof. □

Proof of Theorem 4.2

First we give Lemma 5, which enables us to add rules of a restricted form into a logic program.

Lemma 5. *Let P_1, P_2 be a logic programs such that: For each rule in P_2 the head of the rule is not in $Atoms(P_1)$, and all its body literals are in $Atoms(P_1)$. Then:*

- *If Δ is a stable model of P_1, then $\Delta' = \Delta \cup S(\Delta)$ is the unique stable model of $P_1 \cup P_2$, such that for all $\mathtt{a} \in Atoms(P_1)$: $\mathtt{a} \in \Delta'$ iff $\mathtt{a} \in \Delta$, where*

$$S(\Delta) = \{\mathtt{h} \mid \mathtt{h} \leftarrow \mathtt{a_1}, \ldots, \mathtt{a_n}, not\,(\mathtt{b_1}), \ldots, not\,(\mathtt{b_m}) \in P_2, \text{ such that } \mathtt{a_i} \in \Delta \text{ for } \mathtt{i} = 1, \ldots, \mathtt{n} \text{ and } \mathtt{b_j} \notin \Delta \text{ for } \mathtt{j} = 1, \ldots, \mathtt{m}\}.$$

- *$P_1 \cup P_2$ has the same number of stable models as P_1.*

Proof:
Let Δ be any stable model of P_1, and $r = \mathtt{h} \leftarrow \mathtt{a_1}, \ldots, \mathtt{a_n}, not\,(\mathtt{b_1}), \ldots, not\,(\mathtt{b_m})$ be any rule of the program P_2. If we create a program $P_1' = P_1 \cup \{r\}$, then clearly P_1' has the stable model $\Delta' = \Delta \cup \{h\}$ iff $\mathtt{a_i} \in \Delta$ for $\mathtt{i} = 1, \ldots, \mathtt{n}$ and $\mathtt{b_j} \notin \Delta$ for $\mathtt{j} = 1, \ldots, \mathtt{m}$, and $\Delta' = \Delta$ otherwise. Also the possible addition of the atom h into the program P_1' does not effect the reduct i.e. $R\,(P_1', not(\overline{\Delta'})) = R\,(P_1', not(\overline{\Delta}))$, because h doesn't appear as a body literal in any rule in $P_1 \cup P_2$. Therefore the number of stable models remains the same after the addition of r. The claim can be now proved by induction on the number of rules added from P_2. □

Corollary 2. *Let P_1 and P_2 be two programs satisfying the requirements of Lemma 5. If Δ is a stable model of $P_1 \cup P_2$, then $\Delta' = \{a \in \Delta \mid Atoms(P_1)\}$ is a stable model of P_1.*

Next we start using Lemma 5 to incrementally prove out translation correct.

Lemma 6. *Let program $P_C(\beta)$ be a program made by adding rules of the type 1 of Def. 4.4 to the program $P_B(\beta)$. The program $P_C(\beta)$ has the same number of stable models as $P_B(\beta)$, and the stable models agree on the set of atoms $Atoms(P_B(\beta))$.*

Proof:
The proof is immediate by Lemma 5. □

Lemma 7. *For any stable model Δ of $P_C(\beta)$ and any $\mathtt{b_i} \in Atoms(P_C(\beta))$ it holds that $\mathtt{b_i} \in \Delta$ iff $b_i \in Cut(Events(\Delta))$.*

Proof:
Fix any stable model Δ of $P_C(\beta)$, and any atom $\mathtt{b_i}$ of the program. Now the rule corresponding to this atom is: $\mathtt{b_i} \leftarrow \mathtt{e_l}$, *not* $(\mathtt{e_{p_1}}), \ldots,$ *not* $(\mathtt{e_{p_n}})$, such that $\{e_l\} = {}^\bullet b_i$, and $\{e_{p_1}, \ldots, e_{p_n}\} = b_i{}^\bullet \setminus CutOffs(\beta)$. The corresponding rule will be in the reduct (and by Lemma 5: $\mathtt{b_i} \in \Delta$) iff $\mathtt{e_l} \in \Delta$ and $\mathtt{e_{p_1}}, \ldots, \mathtt{e_{p_n}} \in \overline{\Delta}$, and thus by Theorem 4.1 $e_l \in Events(\Delta)$ and $e_{p_1}, \ldots, e_{p_n} \notin Events(\Delta)$, which is exactly the case when $b_i \in Cut(Events(\Delta))$. □

Lemma 8. *Let $P_D(\beta)$ be the logic program translation of the prefix β by Def. 4.4, and let Δ be a stable model of $P_D(\beta)$. Then the set of events $Events(\Delta)$ is a deadlock configuration of β. Additionally $P_D(\beta)$ has the same number of stable models as there are deadlock configurations of β, which contain no cut-off events.*

Proof:
The program $P_D(\beta)$ is the program $P_C(\beta)$ with only integrity rules of the type 2 of Def. 4.4 added. Thus the set of stable models of $P_D(\beta)$ is a subset of the stable models of the program $P_C(\beta)$, which by Lemma 6 correspond to the configurations of β which contain no cut-off events. Fix any stable model Δ of $P_C(\beta)$. There are now two cases to consider:

- $Events(\Delta)$ is not a deadlock configuration of β: Thus there must exist an event e_i (cut-off or not) which is enabled by $Cut(Events(\Delta))$. Consider now the rule of type 2 corresponding to the event e_i of the form: $\leftarrow \mathtt{b_{p_1}}, \ldots, \mathtt{b_{p_n}}$, such that $\{b_{p_1}, \ldots, b_{p_n}\} = {}^\bullet e_i$. Now by Lemma 7 each of the atoms $\mathtt{b_{p_j}} \in \Delta$ iff $b_{p_j} \in Cut(Events(\Delta))$. Thus the integrity rule for the event e_i will be used, which implies Δ is not a stable model of $P_D(\beta)$.
- $Events(\Delta)$ is a deadlock configuration of β: Thus there is no event e_i (cut-off or not) which is enabled by $Cut(Events(\Delta))$. Now by Lemma 7 each of the atoms $\mathtt{b_i} \in \Delta$ iff $b_i \in Cut(Events(\Delta))$. Therefore none of the rules of the type 2 of Def. 4.4 can be used, which implies Δ is a stable model of $P_D(\beta)$.

We have now found a one-to-one correspondence between the the stable models of $P_D(\beta)$ and the deadlock configurations of β which contain no cut-off events. □

Now we have all the ingredients needed to prove Theorem 4.2. The fact that β is a finite complete prefix of a 1-safe net system Σ guarantees the following. For each reachable marking M of Σ there exists a configuration C of β with no cut-off events, such that $Mark(C) = M$, and for every transition t enabled in M there exists a configuration $C \cup \{e\}$ such that $e \notin C$ and $h(e) = t$. Clearly this also holds for all reachable deadlock markings. The finite complete prefix β will thus have a configuration C with no cut-off events, which can not be extended by any event $e \in E$, iff Σ has a reachable deadlock marking. Now Lemma 8 has shown a one-to-one correspondence between deadlock configurations without cut-off events, and stable models of $P_D(\beta)$. Therefore $P_D(\beta)$ will have a stable model iff Σ has a reachable deadlock marking. It also holds by Lemma 8 that for any stable model Δ of $P_D(\beta)$, $C = Events(\Delta)$ is a deadlock configuration of β, such that $Mark(C)$ a reachable deadlock marking of Σ. □

Proof of Theorem 4.3

We prove the Theorem 4.3 by stepwise adding rules to the base program $P_B(\beta)$.

Lemma 9. *Let program $P_P(\beta, \phi)$ be a program made by adding rules of the type 1 of Def. 4.6 to the program $P_B(\beta)$. The program $P_P(\beta, \phi)$ has the same number of stable models as $P_B(\beta)$, and the stable models agree on the set of atoms $Atoms(P_B(\beta))$.*

Proof:
The proof is immediate by Lemma 5. □

Lemma 10. *For any stable model Δ of $P_P(\beta, \phi)$ and for any $\mathtt{b_i} \in Atoms(P_P(\beta, \phi))$ it holds that $\mathtt{b_i} \in \Delta$ iff $b_i \in Cut(Events(\Delta))$.*

Proof:
Identical to the proof of Lemma 7 when $P_C(\beta)$ is replaced by $P_P(\beta, \phi)$. □

Lemma 11. *Let program $P_Q(\beta, \phi)$ be a program made by adding rules of the type 2 of Def. 4.6 to the program $P_P(\beta, \phi)$. The program $P_Q(\beta, \phi)$ has the same number of stable models as $P_P(\beta, \phi)$, and the stable models agree on the set of atoms $Atoms(P_P(\beta, \phi))$.*

Proof:
The proof is immediate by Lemma 5. □

Lemma 12. *For any stable model Δ of $P_Q(\beta, \phi)$ and for any $\mathtt{s_i} \in Atoms(P_Q(\beta, \phi))$ it holds that $\mathtt{s_i} \in \Delta$ iff $s_i \in Mark(Events(\Delta))$.*

Proof:
Fix any stable model Δ of $P_Q(\beta, \phi)$, and any atom $\mathtt{s_i}$ of the program. Now the rules corresponding to this atom are all of the form: $\mathtt{s_i} \leftarrow \mathtt{b_i}$, such that $s_i = h(b_i)$. Now clearly by Lemma 5: $\mathtt{s_i} \in \Delta$ iff $\mathtt{b_i} \in \Delta$ for some condition b_i for which $s_i = h(b_i)$, which combined with Lemma 10 implies the claim. □

We can now prove Theorem 4.3. The fact that β is a finite complete prefix of a 1-safe net system Σ guarantees the following. For each reachable marking M of Σ there exists a configuration C of β with no cut-off events, such that $\mathit{Mark}(C) = M$. The stable models of the program $P_R(\beta, \phi)$ are always a subset of the stable models of the program $P_Q(\beta, \phi)$, because only integrity rules of type 3 and 4 of Def. 4.6 have been added. By Lemma 11 there exists a stable model of $P_Q(\beta, \phi)$ corresponding to each configuration of the prefix β which contains no cut-off events. Also Lemma 12 shows that the atoms $\mathtt{s_i}$ reflect the corresponding marking of Σ. There are now two cases left to prove:

- If Σ has a reachable marking M which agrees with ϕ, then by Lemma 11 there exists a stable model Δ of $P_Q(\beta, \phi)$, such that $\mathit{Mark}(\mathit{Events}(\Delta)) = M$. Because M agrees with ϕ, it holds by Lemma 12 that for all $\{\mathtt{s_i} \mid s_i \in S^+\}$: $\mathtt{s_i} \in \Delta$, and also for all $\{\mathtt{s_j} \mid s_j \in S^-\}$: $\mathtt{s_j} \in \overline{\Delta}$. Therefore there is no integrity rule in $P_R(\beta, \phi)$ which can be used, which implies Δ is also a stable model of $P_R(\beta, \phi)$.
- If Σ has a reachable marking M which does not agree with ϕ, then by Lemma 11 there exists a stable model Δ of $P_Q(\beta, \phi)$, such that $\mathit{Mark}(\mathit{Events}(\Delta)) = M$. Because M does not agree with ϕ, it holds by Lemma 12 that either there exists a place s_i in S^+ such that $\mathtt{s_i} \in \overline{\Delta}$, or there exists a place s_j in S^- such that $\mathtt{s_j} \in \Delta$. In the first case an integrity rule of type 3, and in the second case an integrity rule of type 4 implies that Δ is not a stable model of $P_R(\beta, \phi)$.

This concludes the proof of Theorem 4.3. □