*Bounded LTL Model Checking with Stable Models**

KEIJO HELJANKO† and ILKKA NIEMELÄ‡
Helsinki University of Technology
Department of Computer Science and Engineering
Laboratory for Theoretical Computer Science
P.O. Box 5400, FIN-02015 HUT, Finland
(*e-mail:* {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. As the model of asynchronous systems a generalisation of communicating automata, 1-safe Petri nets, are used. It is shown how a 1-safe Petri net and a requirement on the behaviour of the net can be translated into a logic program such that the bounded model checking problem for the net can be solved by computing stable models of the corresponding program. The use of the stable model semantics leads to compact encodings of bounded reachability and deadlock detection tasks as well as the more general problem of bounded model checking of linear temporal logic. Correctness proofs of the devised translations are given, and some experimental results using the translation and the `Smodels` system are presented.

KEYWORDS: bounded model checking, stable models, LTL, step semantics

1 Introduction

Recently, a novel paradigm for applying declarative logic programming techniques has been proposed. In this approach, called *answer set programming* (a term coined by Vladimir Lifschitz), a problem is solved by devising a logic program such that models of the program provide the answers to the problem (Lif99; MT99; Nie99). Much of this work has been based on the stable model semantics (GL88) and there are efficient systems `DLV` (`http://www.dbai.tuwien.ac.at/proj/dlv/`) and `Smodels` (`http://www.tcs.hut.fi/Software/smodels/`) for computing stable models of logic programs. Using such an answer set programming system a problem is solved by writing a logic program whose stable models capture the solutions of the problem and then employing the system to compute a solution, i.e., a stable model.

* This is an extended version of a paper titled "Bounded LTL Model Checking with Stable Models" presented at the 6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'2001), Vienna, Austria, September 2001.
† The financial support of Academy of Finland (Projects 53695, 47754) and Foundation of Technology (Tekniikan Edistämissäätiö), Helsinki, Finland are gratefully acknowledged.
‡ The financial support of Academy of Finland (Projects 53695, 47754) is gratefully acknowledged.

In this paper we put forward symbolic model checking (BCM$^+$92; CGP99) 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 reachable states of the system. Tools based on this approach (with various enhancements) include, e.g., the SPIN system (Hol97), which supports extended finite state machines communicating through FIFO queues, and the Petri net model based PROD tool (VHL97). The main problem with enumerative model checkers is the amount of memory needed for 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. Bounded model checking (BCCZ99) has been proposed as a technique for overcoming the space problem by replacing BDDs with satisfiability (SAT) checking techniques because typical SAT checkers use only polynomial amount of memory. The idea is roughly the following. Given a sequential digital circuit, a (temporal) property to be verified, and a bound n, the behaviour 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 $\overline{R}$. The translation to Boolean formulas is done so that $S \wedge \overline{R}$ is satisfiable iff the system has a behaviour 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.

Until now bounded model checking has been applied to synchronous hardware verification and little attention has been given to knowledge representation issues such as developing concise and efficient logical representation of system behaviour. In this work we study the knowledge representation problem and employ ideas used in reducing planning to stable model computation (Nie99). The aim is to develop techniques such that the behaviour of an asynchronous concurrent system can be encoded compactly and the inherent concurrency in the system could be exploited in model checking the system. To illustrate the approach we use a simple basic Petri net model of asynchronous systems, 1-safe Place/Transition nets (P/T nets), which is an interesting generalisation of communicating automata (DR98). Thus properties of finite state systems composed of finite state machine components can be verified using model checkers for 1-safe Petri nets.

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 develop a compact encoding of bounded model checking as the problem of finding stable models of logic programs. We first show how to treat reachability properties such as

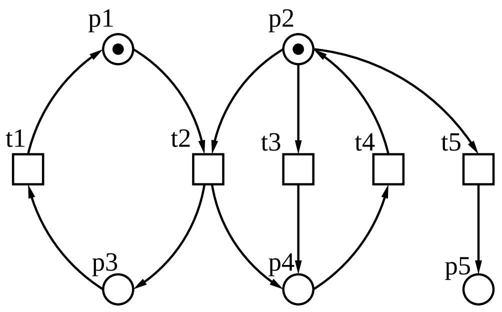

Fig. 1. Running Example

deadlocks and then demonstrate how to extend the approach to cope with properties expressed in linear temporal logic (LTL). We discuss initial experimental results and end with some concluding remarks.

2 Petri nets and bounded model checking

There are several Petri net derived models presented in the literature. We will use P/T-nets which are one of the simplest forms of Petri nets. We will use as a running example the P/T-net presented in Fig. 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 by 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 $\langle P, T, F\rangle$ called the *initial marking*. A marking is graphically denoted by a distribution of tokens on the places of the net. In our running example in Fig. 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 $t2$ is enabled in the initial marking M_0, and thus $M_0 \xrightarrow{t2} M'$, where $M' = \{p3, p4\}$. A marking M is a *deadlock* if no transition $t \in T$ is enabled by M. In our running example the marking $M = \{p1, p5\}$ is a deadlock.

A marking M_n is *reachable* in Σ if there is an *execution*, i.e., a (possibly empty) sequence of transitions $t_0, t_1, \ldots, t_{n-1}$ and markings $M_1, M_2, \ldots, M_{n-1}$ such that: $M_0 \xrightarrow{t_0} M_1 \xrightarrow{t_1} \ldots M_{n-1} \xrightarrow{t_{n-1}} M_n$. A marking M is reachable within a bound n, if there is an execution with at most n transitions, with which M is reachable.

The net system may also have *infinite executions*, i.e., infinite sequences of transitions $t_0, t_1, \ldots$ and markings $M_1, M_2, \ldots$ such that: $M_0 \xrightarrow{t_0} M_1 \xrightarrow{t_1} \ldots$. The *maximal executions* of a net system are the infinite executions of the net system together with the (finite) executions leading to a deadlock marking.

A marking M is 1-safe if $\forall p \in P : M(p) \leq 1$. A P/T-net is 1-safe if all its reachable markings are 1-safe. We will restrict ourselves to finite P/T-nets which are 1-safe, 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, it can be fired 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 interleavings 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 a step.

We say that a marking M_n is *reachable in step semantics* in a 1-safe P/T-net if there is a *step execution*, i.e., a (possibly empty) sequences $S_0, S_1, \ldots, S_{n-1}$ of steps and $M_1, M_2, \ldots, M_{n-1}$ of markings such that: $M_0 \xrightarrow{S_0} M_1 \xrightarrow{S_1} \ldots M_{n-1} \xrightarrow{S_{n-1}} M_n$. 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. We will refer to the "normal semantics" as *interleaving semantics*. The *infinite step executions* and *maximal step executions* are defined in a similar way as in the interleaving case.

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 which implicitly follows from the results of (BD87).

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

Linear temporal logic (LTL). The linear temporal logic LTL is one of the most widely used logics for model checking reactive systems, see e.g., (CGP99). The basic idea is to specify properties that the system should have using LTL. A model checker is then used to check whether all behaviours of the system are models of the specification formula. If not, then the model checker outputs a behaviour of the system which violates the given specification.

Given a finite set AP of atomic propositions, the syntax of LTL is given by:

$$\varphi ::= p \in AP \mid \neg\varphi_1 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \, U \, \varphi_2 \mid \varphi_1 \, R \, \varphi_2 \; .$$

Note that we do not define the often used next-time operator $X \, \varphi_1$. This is a commonly used tradeoff which in our case allows the combination of the step semantics with LTL model checking.

We use $V = 2^{AP}$ as our alphabet. We denote by V^+ all finite sequences over V excluding the empty sequence, and with V^ω all infinite sequences over V. A word $w \in V^+ \cup V^\omega$ is thus either a finite sequence $w = x_0\, x_1\, \ldots\, x_n$ or an infinite sequence $w = x_0\, x_1\, \ldots$, such that $x_i \in V$ for all $i \geq 0$. For a word w we define $w_{(i)} = x_i$, and denote by $w^{(i)}$ the suffix of w starting at x_i. When $w \in V^+$ we define $|w|$ to be the length of the word w, and in the case $w \in V^\omega$ we define $|w| = \omega$ where ω is greater than any natural number.

The relation $w \models \varphi$ is defined inductively as follows:

- $w \models p$ iff $p \in w_{(0)}$ for $p \in AP$,
- $w \models \neg\varphi_1$ iff not $w \models \varphi_1$,
- $w \models \varphi_1 \vee \varphi_2$ iff $w \models \varphi_1$ or $w \models \varphi_2$,
- $w \models \varphi_1 \wedge \varphi_2$ iff $w \models \varphi_1$ and $w \models \varphi_2$,
- $w \models \varphi_1\, U\, \varphi_2$ iff there exists $0 \leq j < |w|$, such that $w^{(j)} \models \varphi_2$ and for all $0 \leq i < j$, $w^{(i)} \models \varphi_1$,
- $w \models \varphi_1\, R\, \varphi_2$ iff for all $0 \leq j < |w|$, if for every $0 \leq i < j$ $w^{(i)} \not\models \varphi_1$ then $w^{(j)} \models \varphi_2$.

We define some shorthand LTL formulas: $\top \equiv p \vee \neg p$ for some arbitrary fixed $p \in AP$, $\bot \equiv \neg\top$, $\Diamond\varphi \equiv (\top\, U\, \varphi)$, $\Box\varphi \equiv (\bot\, R\, \varphi)$, and $\varphi_1 \rightarrow \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$. The temporal operators are called: U for "until", R for "release", $\Diamond$ for "eventually", and $\Box$ for "globally". Our definition of the semantics of LTL above is somewhat redundant. This was done on purpose, as we often in this work use LTL formulas in *positive normal form*, in which only a restricted use of negations is allowed. To be more specific, an LTL formula is said to be in positive normal form when all negations in the formula appear directly before an atomic proposition. A formula can be put into positive normal form with the following equivalences (and their duals): $\neg\neg\varphi \equiv \varphi$, $\neg(\varphi_1 \vee \varphi_2) \equiv \neg\varphi_1 \wedge \neg\varphi_2$, and $\neg(\varphi_1\, U\, \varphi_2) \equiv \neg\varphi_1\, R\, \neg\varphi_2$. Note that converting a formula into positive normal form does not involve a blowup.

Some examples of practical use of LTL formulas are: $\Box\neg(cs_1 \wedge cs_2)$ (it always holds that two processes are not at the same time in a critical section), $\Box(req \rightarrow \Diamond ack)$ (it is always the case that a request is eventually followed by an acknowledgement), and $((\Box\Diamond sch_1) \wedge (\Box\Diamond sch_2)) \rightarrow (\Box(tr_1 \rightarrow \Diamond cs_1))$ (if both process 1 and 2 are scheduled infinitely often, then always the entering of process 1 in the trying section is followed by the process 1 eventually entering the critical section).

Given a 1-safe P/T net Σ, we use a chosen subset of the places as the atomic propositions AP. A maximal (interleaving) execution $M_0 \xrightarrow{t_0} M_1 \xrightarrow{t_1} \ldots$ satisfies φ iff the corresponding word $w = (M_0 \cap AP), (M_1 \cap AP), \ldots$ satisfies φ. We say that Σ satisfies φ iff every maximal execution starting from the initial marking M_0 satisfies φ. Alternatively, Σ does not satisfy φ if there exists a maximal execution starting from M_0 which satisfies $\neg\varphi$. We call such an execution a *counterexample.* Notice that we restrict ourselves to maximal executions and thus our counterexamples are either infinite executions or finite executions leading to a deadlock (recall the definition of maximal executions).

The temporal logic LTL can specify quite complex properties of reactive systems. In many cases it suffices to reason about much simpler temporal properties. A typical example is the reachability of a marking satisfying some condition C which in the LTL setting corresponds to finding a counterexample for a formula $\Box\neg C$. An important reachability based problem is deadlock detection.

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

Most analysis questions including deadlock detection and LTL model checking are PSPACE-complete in the size of a 1-safe Petri net, see e.g., (Esp98). In *bounded model checking* we fix a bound n and look for counterexamples which are shorter than the given bound n. For example, in the case of *bounded deadlock detection* we look for executions reaching a deadlock in at most n transitions. It is easy to show that, e.g., the bounded deadlock detection problem is NP-complete (when the bound n is given in unary coding). This idea can also be applied to LTL model checking. In (BCCZ99) *bounded LTL model checking* is introduced. They also discuss how to ensure that a given bound n is sufficient to guarantee completeness. Unfortunately, getting an exact bound is often computationally infeasible, and easily obtainable upper bounds are too large. In the case of 1-safe P/T-nets they are exponential in the number of places in the net. Therefore the bounded model checking results are usually not conclusive if a counterexample is not found. Thus bounded model checking is at its best in "bug hunting", and not as easily applicable in verifying systems to be correct.

3 From bounded model checking to answer set programming

In this section we show how to solve bounded LTL model checking problems using answer set programming based on normal logic programs with the stable model semantics. The basic idea is to reduce a bounded model checking problem to a stable model computation task, i.e., to devise for a P/T-net, a bound, and a temporal property to be checked a logic program such that the stable models of the program correspond directly to executions of the net within the bound violating the property. Then an implementation of the stable model semantics can be used to perform bounded model checking tasks. First we briefly review the stable model semantics (GL88) and discuss a couple of useful shorthands to be used in the encodings as well as the basis of an answer set programming methodology with rules. Then we address the encoding of checking reachability properties and finally extend the approach to handle full LTL model checking.

3.1 Stable model semantics

For encoding bounded model checking problems we use normal logic programs with stable model semantics (GL88). A normal rule is of the form

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

where each a, b_i, c_j is a ground atom. Models of a program are sets of ground atoms. A set of atoms Δ is said to satisfy an atom a if $a \in \Delta$ and a negative literal not a if $a \notin \Delta$. A rule r of the form (1) is satisfied by Δ if the head a is satisfied whenever every body literal $b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n$ is satisfied by Δ and a program Π is satisfied by Δ if each rule in Π is satisfied by Δ (denoted $\Delta \models \Pi$).

Stable models of a program are sets of ground atoms which satisfy all the rules of the program and are justified by the rules. This is captured using the concept of a *reduct*. For a program Π and a set of atoms Δ, the reduct Π^Δ is defined by

$$\Pi^\Delta = \{a \leftarrow b_1, \ldots, b_m \mid a \leftarrow b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n \in \Pi, \\ \{c_1, \ldots, c_n\} \cap \Delta = \emptyset\}$$

i.e., a reduct Π^Δ does not contain any negative literals and, hence, has a unique subset minimal set of atoms satisfying it.

Definition 2
A set of atoms Δ is a stable model of a program Π iff Δ is the unique minimal set of atoms satisfying Π^Δ.

We employ three extensions which can be seen as compact shorthands for normal rules. We use *integrity constraints*, i.e., rules

$$\leftarrow b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n \tag{2}$$

with an empty head. Such a constraint can be taken as a shorthand for a rule

$$f \leftarrow \text{not } f, b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n$$

where f is a new atom. Notice that a stable model Δ satisfies an integrity constraint (2) only if at least one of its body literals is not satisfied by Δ.

For expressing the choice whether to include an atom in a stable model we use *choice rules*. They are normal rules where the head is in brackets with the idea that the head can be included in a stable model only if the body holds but it can be left out, too. Such a construct can be represented using normal rules by introducing a new atom. For example, the choice rule on the left corresponds to the two normal rules on the right where a' is a new atom.

$$\{a\} \leftarrow b, \text{not } c \qquad \leadsto \qquad \begin{array}{l} a \leftarrow \text{not } a', b, \text{not } c \\ a' \leftarrow \text{not } a \end{array}$$

Finally, a compact encoding of *conflicts* is needed, i.e., rules of the form

$$\leftarrow 2\{a_1, \ldots, a_n\} \tag{3}$$

saying that a stable model cannot contain any two atoms out of a set of atoms $\{a_1, \ldots, a_n\}$. Such a rule can be expressed, e.g., by adding a rule $f \leftarrow \text{not } f, a_i, a_j$, where f is a new atom, for each pair a_i, a_j from $\{a_1, \ldots, a_n\}$, i.e., using $\mathcal{O}(n^2)$ rules. Choice and conflict rules are simple cases of cardinality constraint rules (NS00). The `Smodels` system provides an implementation for cardinality constraint rules and includes primitives supporting directly such constraints without translating them first to corresponding normal rules.

A straightforward method of using logic program rules for answer set programming can be based on a *generate and test* idea. A set of rules plays the role of a generator capturing stable models corresponding to all candidate solutions and another set of rules, testers, eliminate the non-valid ones. A systematic way of using this method can be based on some simple modularity properties of stable model semantics which are given below as propositions where the first two are straightforward consequences of the splitting theorem (LT94).

The propositions play an important role in proving the correctness of our logic program encodings. The first one says that if rules defining new atoms are added, then a stable model of the original program can be obtained directly from a stable model of the extended program. Often a tester is encoded using a stratified set of rules and an integrity constraint. The next two propositions show that this does not introduce new stable models but extends the original ones and possibly eliminates some of them.

Proposition 1
Let Π_1 and Π_2 be programs such that the atoms in the heads of the rules in Π_2 do not occur in Π_1. Then for every stable model Δ of $\Pi_1 \cup \Pi_2$, $\Delta \cap \text{Atoms}(\Pi_1)$ is a stable model of Π_1 where $\text{Atoms}(\Pi_1)$ denotes the set of atoms appearing in Π_1.

Proposition 2
Let Π_1 be a program and Π_2 a stratified program such that the atoms in the heads of the rules in Π_2 do not occur in Π_1. Then for every stable model Δ_1 of Π_1 there is a unique stable model Δ of $\Pi_1 \cup \Pi_2$ such that $\Delta_1 = \Delta \cap \text{Atoms}(\Pi_1)$.

Proposition 3
Let Π be a program. Then Δ is a stable model of Π and satisfies an integrity constraint ic (2) iff Δ is a stable model of $\Pi \cup \{ic\}$.

3.2 Reachability checking

Now we devise a method for translating bounded reachability problems of 1-safe P/T-nets to tasks of finding stable models. Consider a net $N = \langle P, T, F \rangle$ and a step bound $n \geq 1$. 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
$$\{p(0)\} \leftarrow \ . \tag{4}$$
- For each transition $t \in T$, and for all $i = 0, 1, \ldots, n-1$, include a rule
$$\{t(i)\} \leftarrow p_1(i), \ldots, p_l(i) \tag{5}$$
where $\{p_1, \ldots, p_l\}$ is the preset of t. Hence, a stable model can contain a transition instance in step i only if its preset holds at step i.

- For each place $p \in P$, for each transition t in the preset of p, and for all $i = 0, 1, \ldots, n-1$, include a rule
$$p(i+1) \leftarrow t(i)\ . \tag{6}$$
These say 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 $i = 0, 1, \ldots, n-1$, if the cardinality of the postset $\{t_1, \ldots, t_l\}$ of p is a least 2, include a rule
$$\leftarrow 2\{t_1(i), \ldots, t_l(i)\}\ . \tag{7}$$
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 $i = 0, 1, \ldots, n-1$,
$$p(i+1) \leftarrow p(i), \text{not } t_1(i), \ldots, \text{not } t_l(i) \tag{8}$$
where $\{t_1, \ldots, t_l\}$ is the set of transitions having p in their preset. This is the *frame axiom* for p stating that p continues to hold if no transition using it occurs.
- Disallow execution of transitions followed by idling. For all $i = 0, 1, \ldots, n-1$, include rules
$$\mathit{idle}(i) \leftarrow \text{not } t_1(i), \ldots, \text{not } t_k(i) \qquad \leftarrow \mathit{idle}(i+1), \text{not } \mathit{idle}(i) \tag{9}$$
where $\{t_1, \ldots, t_k\} = T$, i.e., the set of all transitions. These rules force all idling to happen at the beginning, followed by non-idling time-steps (if any).

As an example consider net N in Fig. 1 for which program $\Pi_{\mathrm{A}}(N, n)$ is given in Fig. 2.

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

Fig. 2. Program $\Pi_{\mathrm{A}}(N, n)$

In $\Pi_{\mathrm{A}}(N, n)$ the initial marking is not constrained. Next we show how to limit markings using rules, i.e., how to construct a set of rules $\Pi_{\mathrm{M}}(C, i)$ that eliminates all stable models which do not satisfy a given Boolean expression C of marking conditions at step i. The set $\Pi_{\mathrm{M}}(C, i)$ includes the rule $\leftarrow$ not $c(i)$ and a set of rules defining $c(i)$ by a systematic translation of the condition C at step i as explained next. A Boolean expression C can be encoded with rules by introducing for each non-atomic subexpression of C a new atom together with rules capturing the conditions under which the subexpression is satisfied in the following way (NS00). Given a Boolean expression C with connectives $\neg, \vee, \wedge$, every subexpression of C of the form $\neg\phi$ is mapped to a rule $c_{\neg\phi} \leftarrow$ not c_ϕ; a subexpression $\phi \wedge \psi$ is mapped to $c_{\phi\wedge\psi} \leftarrow c_\phi, c_\psi$ and $\phi \vee \psi$ to the two rules $c_{\phi\vee\psi} \leftarrow c_\phi$ and $c_{\phi\vee\psi} \leftarrow c_\psi$ where c_ψ, c_ϕ are new atoms introduced for the non-atomic subexpressions. These are not needed for the atomic ones, i.e., $c_a = a$ for an atom a. The conditions for a step i are then obtained by indexing all atoms with i.

The encoding of marking conditions is illustrated by considering a condition $C : p_1 \wedge (\neg p_2 \vee p_3)$ saying that $p_1 \in M$ and ($p_2 \notin M$ or $p_3 \in M$) and a step i. Now the set of rules $\Pi_{\mathrm{M}}(C, i)$ is

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

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.

Proof
See Appendix A.1. □

The deadlock detection problem is now just a special case of a reachability property where the rules $\Pi_{\mathrm{M}}(C, n)$ exclude markings with some transition enabled. This set of rules is denoted by $\Pi_{\mathrm{D}}(N, n)$ and it consists of the rule $\leftarrow$ *live* and the program $\Pi_{\mathrm{L}}(N, n)$ which includes for each transition $t \in T$ and its preset $\{p_1, \ldots, p_l\}$, a rule

$$\mathit{live} \leftarrow p_1(n), \ldots, p_l(n) \ . \qquad (10)$$

For our running example, the rules $\Pi_{\mathrm{L}}(N, n)$ are

$$\mathit{live} \leftarrow p3(n) \qquad \mathit{live} \leftarrow p1(n), p2(n) \qquad \mathit{live} \leftarrow p2(n) \qquad \mathit{live} \leftarrow p4(n) \ .$$

3.3 Bounded LTL model checking

Our strategy for finding counterexamples for LTL formula φ (i.e., executions satisfying $\neg\varphi$) is almost the same as in (BCCZ99). The main difference is that we allow the system under model checking to have reachable deadlocks, while their translation does not allow this. This is also a difference to our previous work (HN01).

Our counterexamples have two basic shapes. On the left in Fig. 3 is a *loop counterexample*, and on the right is a *counterexample without loop*. Loop counterexam-

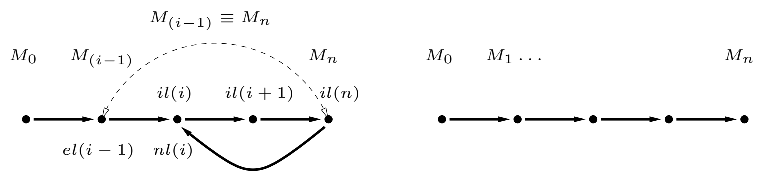

Fig. 3. Two counterexample possibilities

ples specify an infinite execution, while counterexamples without a loop specify a finite execution. The arcs of the figure denote the "next state" of each state. Notice in the loop counterexample that if $M_{(i-1)}$ is equivalent to the last state M_n, the state M_i is the "next state" of M_n. The counterexamples without loop can additionally be divided into *deadlock executions* (ending in a deadlock state), and *non-maximal executions* (ending in a state which is not a deadlock).

In the case of non-maximal executions our encoding is a cautious one, and we will find counterexamples which exist, no matter how the non-maximal execution is extended into a maximal one. (Recall that we have defined the semantics of LTL over maximal executions of the net system.) Finding non-maximal counterexample executions is in fact only an optimisation. It was introduced in (BCCZ99), and allows some counterexamples to be found with smaller bounds than would otherwise be possible.

In the encoding we use the auxiliary atoms $el(i), le, nl(i), il(i)$ with following intuition (see Fig. 3 for an example). The $el(i)$ atom is in a stable model for the state i that is equivalent with the last state n and le is in the model if a loop exists, i.e., some $el(i)$ is in the model. The $nl(i)$ atom is in a model for the "next state" i of the last state, while $il(i)$ is in the model for all states i in the loop.

Given an LTL formula f in positive normal form[1] (when the formula to be model checked is φ, the formula f is equivalent to $\neg\varphi$ with negations pushed in), and a bound $n \geq 1$ we construct a program $\Pi_{\mathrm{LTL}}(f, n)$ as follows.

- Guess which state is equivalent to the last (if any). For all $0 \leq i \leq n-1$ add rule
$$\{el(i)\} \leftarrow \ . \tag{11}$$
- Disallow guessing two or more. (Guessing none is allowed though.) Add rule
$$\leftarrow 2\{el(0), el(1), \ldots, el(n-1)\}\ . \tag{12}$$
- Check that the guess is correct. For all $0 \leq i \leq n-1$, $p \in P$ include rules
$$\leftarrow el(i), p(i), \mathrm{not}\ p(n) \qquad \leftarrow el(i), p(n), \mathrm{not}\ p(i)\ . \tag{13}$$

[1] Using the positive normal form is required to handle non-maximal counterexample executions, for which the duality $f_1\, R\, f_2 \equiv \neg(\neg f_1\, U\, \neg f_2)$ can not be used, see (BCCZ99).

Formula type	Translation	Formula type	Translation
p, for $p \in AP$	$f(i) \leftarrow p(i)$	$\neg p$, for $p \in AP$	$f(i) \leftarrow \text{not } p(i)$
$f_1 \vee f_2$	$f(i) \leftarrow f_1(i)$ $f(i) \leftarrow f_2(i)$	$f_1 \wedge f_2$	$f(i) \leftarrow f_1(i), f_2(i)$
$f_1 \, U \, f_2$	$f(i) \leftarrow f_2(i)$ $f(i) \leftarrow f_1(i), f(i+1)$ $f(n+1) \leftarrow nl(i), f(i)$	$f_1 \, R \, f_2$	$f(i) \leftarrow f_2(i), f_1(i)$ $f(i) \leftarrow f_2(i), f(i+1)$ $f(n+1) \leftarrow nl(i), f(i)$ $f(n+1) \leftarrow le, \text{not } c(f)$ $c(f) \leftarrow il(i), \text{not } f_2(i)$ $f(n) \leftarrow f_2(n), \text{not } live$

Fig. 4. Translation of an LTL formula f

- Specify auxiliary loop related atoms. For all $0 \le i \le n-1$, include rules
$$le \leftarrow el(i) \quad nl(i+1) \leftarrow el(i) \quad il(i+1) \leftarrow el(i) \quad il(i+1) \leftarrow il(i)\ . \tag{14}$$
- Require that if a loop exists, the last step contains a transition to disallow looping by idling. Add the rule
$$\leftarrow le, idle(n-1)\ . \tag{15}$$
- Allow at most one visible transition in a step to eliminate steps which cannot be interleaved to yield a counterexample. For all $0 \le i \le n-1$, add rule
$$\leftarrow 2\{t_1(i), \ldots, t_k(i)\} \tag{16}$$
where $\{t_1, \ldots, t_k\}$ is the set of *visible transitions*, i.e., the transitions whose firing changes the marking of a place p appearing in the formula f. More formally, a transition $t \in T$ is visible, if there exists a place $p \in AP$ such that $F(t,p) - F(p,t) \neq 0$.

We recursively translate the formula f by first translating its subformulas, and then f as follows. For all $0 \le i \le n$, add the rules given by Fig. 4.[2] Finally we require that the top level formula f should hold in the initial marking
$$\leftarrow \text{not } f(0)\ . \tag{17}$$
With this program $\Pi_{\text{LTL}}(f,n)$ we get our main result.

Theorem 3
Let f be an LTL formula in positive normal form and $N = \langle P, T, F \rangle$ be a 1-safe P/T-net for all initial markings satisfying a condition C_0. If $\Pi_{\text{M}}(C_0, 0) \cup \Pi_{\text{A}}(N,n) \cup \Pi_{\text{L}}(N,n) \cup \Pi_{\text{LTL}}(f,n)$ has a stable model, then there is a maximal execution of N from an initial marking satisfying C_0 which satisfies f.

[2] An equivalence explaining the release translation: $f_1 \, R \, f_2 \equiv (f_2 \, U \, (f_1 \wedge f_2)) \vee (\Box f_2)$.

Proof
See Appendix A.2. □

We also have the following completeness result for our translation. First we define the notion of a *looping execution.* A finite execution $M_0 \xrightarrow{t_0} M_1 \xrightarrow{t_1} \dots M_{n-1} \xrightarrow{t_{n-1}} M_n$ is a looping execution, if $n \geq 1$ and there exists an index $l < n$ such that $M_l = M_n$. A looping execution together with the index l is a finite witness to the existence of the corresponding (infinite) maximal execution σ of the net system N which visits the sequence of states $M_0, M_1, \dots, M_l, M_{l+1}, \dots, M_k, M_{l+1}, \dots, M_k, \dots$.

Theorem 4
Let f be an LTL formula in positive normal form and $N = \langle P, T, F \rangle$ be a 1-safe P/T-net for all initial markings satisfying a condition C_0. If N has a looping or deadlock execution of at most length n starting from an initial marking satisfying C_0 such that some corresponding maximal execution σ satisfies f, then $\Pi_{\mathrm{M}}(C_0, 0) \cup \Pi_{\mathrm{A}}(N, n) \cup \Pi_{\mathrm{L}}(N, n) \cup \Pi_{\mathrm{LTL}}(f, n)$ has a stable model.

Proof
See Appendix A.3. □

The size of the program in Theorem 3 is linear in the size of the net and formula, i.e., $\mathcal{O}((|P| + |T| + |F| + |f|) \cdot n)$. The semantics of LTL is defined over interleaving executions. A novelty of the translation is that it allows concurrency between invisible transitions.

We could simplify the LTL translation presented above in following ways. Firstly, if the net system is known to be deadlock free, the release translation in Fig. 4 can be simplified by removing the rule

$$f(n) \leftarrow f_2(n), \mathrm{not}\ \mathit{live},$$

and also the (now unnecessary) subprogram $\Pi_{\mathrm{L}}(N, n)$.

Secondly, if we remove the possibility of obtaining non-maximal counterexample executions, the release translation can be removed fully by using the equivalence $\varphi_1 \, R \, \varphi_2 \equiv \neg \, (\neg \, \varphi_1 \, U \, \neg \, \varphi_2)$ and adding (the obvious) translation for negation. This can not be done when non-maximal counterexamples are used, because the equivalence does not hold in that case. As an example, one can not deduce from the fact that $\neg \Diamond \neg \, \varphi$ holds for a non-maximal execution σ that $\Box \, \varphi$ holds for any maximal execution σ' such that σ is a prefix of σ'. The non-maximal counterexample executions are quite valuable in practice, as using them violations to safety properties can be found with smaller bounds. Therefore we chose to use a more complicated translation for release.

Forcing interleaving semantics. We can create the interleaving semantics versions of bounded model checking problems by adding a set of rules $\Pi_{\mathrm{I}}(N, n)$. It includes for each time step $0 \leq i \leq n - 1$ a rule

$$\leftarrow 2\{t_1(i), \dots, t_m(i)\} \tag{18}$$

where $\{t_1, \dots, t_m\}$ 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 program solving a bounded model checking problem in the step semantics using any of the translations above. Then the program $\Pi_{\mathrm{S}}(N, n) \cup \Pi_{\mathrm{I}}(N, n)$ solves the same problem in the interleaving semantics.

3.4 Relation to previous work

Logic programming techniques have been used to model checking branching time modal logics like the modal mu-calculus and CTL where model checking can be reduced to solving equations with least and greatest fixed points. A state of the art example of this approach is the XMC system (RRS$^+$00) which has been extended to handle also linear temporal logic LTL using the standard tableau style approach (PR00). This method has the disadvantage that the size of the resulting tableau can be exponential w.r.t. the size of the temporal formula to be checked. The exponential worst case space complexity, which is present in typical LTL model checkers, is avoided in bounded model checking where the space complexity remains polynomial also w.r.t. the temporal formula.

In previous work on bounded model checking little attention has been given to the knowledge representation problem of encoding succinctly the unfolded behavior and the temporal property. We address this problem and develop an encoding of the behavior of an asynchronous system which is linear in the size of the system description (Petri net) and the formula as well as in the number of steps.

Our approach extends the previous work in several respects. Earlier research has been based on the interleaving semantics. Our work allows the use of the step semantics which enables the exploitation of the inherent concurrency of the system in model checking. The standard approach (BCCZ99) assumes that the system to be model-checked is deadlock-free while we can do LTL model checking for systems with reachable deadlocks.

We develop a more compact encoding of bounded LTL model checking. Our encoding is linear in the size of the net, the formula and the bound. In (BCCZ99) the encoding is superlinear in the size of the formula. The paper provides no upper bound on the size w.r.t. the formula but states that it is polynomial in the size of the formula if common subexpressions are shared and quadratic in the bound. These same observations can also be made of the optimised version of the translation presented in (CPRS02). The compactness of our encoding is due to the fact that the stable model semantics supports least fixed point evaluation of recursive rules which is exploited in translating the until and release formulas.

For simple temporal properties such as reachability and deadlock detection our approach could be quite directly used as a basis for a similar treatment using propositional logic and satisfiability (SAT) checkers. This is fairly straightforward by using the ideas of Clark's completion and Fages' theorem (Fag94) as our encoding produces acyclic programs except for the choice rules which need a special treatment.

4 Experiments

We have implemented the deadlock detection and LTL model checking translations presented in the previous section in a bounded model checker `boundsmodels 1.0` which uses `Smodels` as the underlying stable model finder. The implementation performs the following optimisations when given a fixed initial marking M_0:

- 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 (7) and (10).

We compare `boundsmodels` to a state of the art model checker `NuSMV 2.1.0` (`http://nusmv.irst.itc.it/`) which contains two different model checking engines (CCG$^+$02). The first one (`NuSMV/BMC`) is a bounded LTL model checker based on the approach of (BCCZ99), and includes some further improvements presented in (CPRS02). It uses as the underlying SAT solver the `zChaff 2001.2.17` (`http://www.ee.princeton.edu/~chaff/`) system (MMZ$^+$01). The second engine (`NuSMV/BDD`) is an efficient implementation of a traditional BDD based model checker.

As benchmarks we use a set of deadlock detection benchmarks collected by Corbett (Cor95), and also hand-crafted LTL model checking problems based on these models. The Corbett models are available both as communicating automata, and in the input language of the `NuSMV` model checker. The communicating automata models were converted to 1-safe P/T-nets by Melzer and Römer (MR97). We use the models which have a deadlock, and are non-trivial to model check.

In deadlock checking experiments for each model and both semantics we increment the used bound until a deadlock is found. We report the time for `Smodels 2.26` to find the first stable model using this bound and the time used by the `NuSMV` model checker. In some cases a model could not be found within a reasonable time (3600 seconds) in which case we report the time used to prove that there is no deadlock within the reported bound.

The deadlock checking experimental results can be found in Table 1. We use "$*$" to denote the fact that `NuSMV` ran out of 900MiB memory limit on DARTES(1) with both engines, so we could not make a comparison in this case. While performing state space size comparisons between Petri net and `NuSMV` models, we found problems in the used communicating automata to Petri net translation, resulting in model differences in ELEV(x) and HART(x). Thus we also excluded these models from comparison denoting this in the table with "-".

The columns are:

- Problem: The problem name with the size of the instance in parenthesis.
- St n: The smallest integer n such that a deadlock could be found using the

Table 1. *Deadlock Checking Experiments*

Problem	St n	St s	Int n	Int s	Bmc n	Bmc s	Bdd s	States
DP(6)	1	0.0	6	0.1	6	0.2	0.1	728
DP(8)	1	0.0	8	2.3	8	2.4	0.1	6560
DP(10)	1	0.0	10	182.5	10	155.9	0.2	59048
DP(12)	1	0.0	>9	707.3	>8	984.4	0.2	531440
KEY(2)	>29	2089.7	>29	2227.8	>30	2531.9	0.1	536
MMGT(3)	7	0.9	10	24.2	10	16.6	0.2	7702
MMGT(4)	8	174.9	12	2533.4	12	84.9	0.4	66308
Q(1)	9	0.0	>17	1051.4	>11	2669.8	2.9	123596
DARTES(1)	32	0.4	32	0.4	∗	∗	∗	>1500000
ELEV(1)	4	0.0	9	0.1	-	-	-	163
ELEV(2)	6	0.2	12	1.8	-	-	-	1092
ELEV(3)	8	1.9	15	94.2	-	-	-	7276
ELEV(4)	10	60.9	>13	656.8	-	-	-	48217
HART(25)	1	0.0	>5	0.4	-	-	-	>1000000
HART(50)	1	0.0	>5	1.7	-	-	-	>1000000
HART(75)	1	0.0	>5	5.1	-	-	-	>1000000
HART(100)	1	0.0	>5	11.6	-	-	-	>1000000

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.
- Bmc n and Bmc s: Same as Int n and Int s above, but for the `NuSMV/BMC` bounded model checking engine.
- Bdd s: Time needed for the `NuSMV/BDD` engine to compute the set of reachable states and to find a state in that set which has no successors.
- States: Number of reachable states of the model (if known).

The time reported is the average of 5 runs where the timing is measured by the `/usr/bin/time` command on a 1GiB RAM, 1GHz AMD (Thunderbird) Athlon PC running Linux. The time needed for creating the `Smodels` input was very small, and therefore omitted.

The `NuSMV/BMC` engine did not directly support deadlock checking, so we had to modify the models slightly to add a proposition *Live* to all the models, which is true iff any transition is enabled. We then ask for counterexamples without a loop for the LTL property $\Box Live$. With the `NuSMV/BDD` engine we use forward reachability checking combined with transition relation totality check limited to the reachable states. The default dynamic variable reordering method is used. We disable for these deadlock checking experiments a time consuming (and unnecessary for deadlock checking) fairness set calculation during `NuSMV/BDD` model initialisation.

Table 2. *LTL Model Checking Experiments*

Problem	St n	St s	Int n	Int s	Bmc n	Bmc s	Bdd s	States
DP(6)	7	0.2	8	0.5	8	4.3	64.8	728
DP(8)	8	1.5	10	5.7	10	64.0	>1800	6560
DP(10)	9	25.9	12	140.1	12	1257.1	>1800	59048
DP(12)	10	889.4	14	>1800	14	>1800	>1800	531440

When comparing our bounded model checker on step and interleaving semantics we note that in many of the experiments the step semantics version finds a deadlock with a smaller bound than the interleaving one. Also, when the bound needed to find the deadlock is fairly small, our bounded model checker is performing well. In the examples ELEV(4), HART(x) and Q(1) we are able to find a counterexample only when using step semantics. In the KEY(2) example we are not 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 is obtained.

When comparing with `NuSMV/BMC` we observe that the step semantics translation is quite competitive, with only `NuSMV/BMC` being better on KEY(2) and MMGT(4). We believe this is mainly due to the smaller bounds obtained using steps. Somewhat surprisingly to us, `NuSMV/BMC` is also worse than interleaving on DP(12) and Q(1). This could be due to either translation or solver differences.

The examples we have used have a small and fairly regular state space. Thus the `NuSMV/BDD` engine is very competitive on them, as expected. The only exception to this rule is DARTES(1), where for some reason the `NuSMV/BDD` engine uses more than 900 MiB of memory. Overall, the results are promising, in particular, for small bounds and the step semantics.

We do not have a large collection of LTL model checking problems available to us. Instead we pick a model family, the dining philosophers problems DP(x), and use a hand-crafted LTL formula for each model. Because the `NuSMV/BDD` LTL model checking engine only works for deadlock free models, we remove all the deadlocks from these examples by making each deadlock state a successor of itself.

The formulas to be checked are hand-crafted to demonstrate potential differences between (BCCZ99) and our proposed method. We study nested until formulas for which the translation of (BCCZ99) seems to be rather complex. In our model the atomic proposition $f_i.up$ has the meaning that fork i is available, and $p_i.eat$ has the meaning that philosopher i is eating. We model check the following formulas. For six philosophers we use the formula:

$$\neg\Box\Diamond(f_5.up\ U\ (p_5.eat \wedge (f_3.up\ U\ (p_3.eat \wedge (f_1.up\ U\ p_1.eat))))),$$

for eight philosophers we use the formula:

$$\neg\Box\Diamond(f_7.up\ U\ (p_7.eat \wedge (f_5.up\ U\ (p_5.eat \wedge (f_3.up\ U\ (p_3.eat \wedge (f_1.up\ U\ p_1.eat))))))),$$

and so on. The counterexample is a model for a formula of the form $\Box\Diamond(\varphi)$, where φ has deeply nested until formulas. Thus in a counterexample φ has to hold infinitely often. As an example, one way to make φ hold in the six philosophers case is to find a state where $(p_5.\mathit{eat} \wedge p_3.\mathit{eat} \wedge p_1.\mathit{eat})$ holds.

The experimental results for the LTL model checking can be found in Table 2. For this set of experiments we use the run time limit of 1800 seconds, and do not try smaller bounds when the limit is exceeded. The columns of the table are as in deadlock checking experiments, except that we are looking for a counterexample to the LTL formula. In these examples `NuSMV` is run with default dynamic BDD variable reordering on.

The experiments show that the step semantics is able to obtain a counterexample for DP(12), while other methods are unable to. The `NuSMV/BMC` engine scales worse than the interleaving semantics translation. By investigating further, we notice that in DP(10) the `zChaff` solver only takes 160.4 seconds, while the generation of the SAT instance for the solver takes almost 1100 seconds. We believe that a large part of this overhead is due to the size of the generated LTL model checking translation. The `NuSMV/BDD` based LTL model checker seems to be scaling worse than for the corresponding deadlock checking examples and it can be observed that the number of BDD operations required for LTL model checking is significantly larger.

The used tools, models, formulas, and logic programs are available at `http://www.tcs.hut.fi/~kepa/experiments/boundsmodels/`.

5 Conclusions

We introduce bounded model checking of asynchronous concurrent systems modelled by 1-safe P/T-nets as an interesting application area for answer set programming. We present mappings from bounded reachability, deadlock detection, and LTL model checking problems of 1-safe P/T-nets to stable model computation. Our approach is capable of doing 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. We handle asynchronous systems using a step semantics whereas previous work on bounded model checking only uses the interleaving semantics (BCCZ99). Furthermore, our encoding is more compact than the previous approach employing propositional satisfiability (BCCZ99). This is because our rule based approach allows to represent executions of the system, e.g. frame axioms, succinctly and supports directly the recursive fixed point computation needed to evaluate LTL formulas. Another feature of our LTL translation is that it does not require the deadlock freeness assumption used by (BCCZ99), and thus we can employ it also with systems which have not been proved deadlock free.

The first experimental results indicate that stable model computation is quite a competitive approach to searching for short executions of the system leading to deadlock and worth further study. More experimental work and comparisons are needed to determine the strength of the approach. In particular, for comparing with SAT checking techniques, it would be interesting to develop a similar treatment of asynchronous systems using a SAT encoding and compare it to the logic program based approach.

Relating the net unfolding method (see (Hel99; MR97) and further references there) to bounded model checking would be interesting. There are also alternative semantics to the two presented in this work (Hel01), applying them to bounded LTL model checking is left for further work.

References

A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99)*, volume 1579 of *Lecture Notes in Computer Science*, pages 193–207. Springer-Verlag, March 1999.

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

E. Best and R. Devillers. Sequential and concurrent behaviour in Petri net theory. *Theoretical Computer Science*, 55(1):87–136, 1987.

A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV 2: An opensource tool for symbolic model checking. In *Proceeding of the 14th International Conference on Computer-Aided Verification (CAV'2002)*, volume 2404 of *Lecture Notes in Computer Science*, pages 359–364. Springer-Verlag, July 2002.

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

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

A. Cimatti, M. Pistore, M. Roveri, and R. Sebastiani. Improving the encoding of LTL model checking into SAT. In *Proceeding of workshop on Verification Model Checking and Abstract Interpretation (VMCAI'2002)*, volume 2294 of *Lecture Notes in Computer Science*, pages 196–207. Springer-Verlag, January 2002.

J. Desel and W. Reisig. Place/Transition Petri nets. In *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*, pages 122–173. Springer-Verlag, 1998.

J. Esparza. Decidability and complexity of Petri net problems – An introduction. In *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*, pages 374–428. Springer-Verlag, 1998.

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

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. The MIT Press, August 1988.

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

K. Heljanko. Bounded reachability checking with process semantics. In *Proceedings of the 12th International Conference on Concurrency Theory (Concur'2001)*, volume 2154 of *Lecture Notes in Computer Science*, pages 218–232. Springer-Verlag, August 2001.

K. Heljanko and I. Niemelä. Bounded LTL model checking with stable models. In *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic*

Reasoning (LPNMR'2001), volume 2173 of *Lecture Notes in Computer Science*, pages 200–212. Springer-Verlag, September 2001.

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

V. Lifschitz. Answer set planning. In *Proceedings of the 16th International Conference on Logic Programming*, pages 25–37. MIT Press, December 1999.

V. Lifschitz and H. Turner. Splitting a logic program. In *Proceedings of the Eleventh International Conference on Logic Programming*, pages 23–37, Cambridge, Mass., 1994. The MIT Press.

M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference (DAC'2001)*, pages 530–535. ACM, June 2001.

S. Melzer and S. Römer. Deadlock checking using net unfoldings. In *Proceeding of 9th International Conference on Computer Aided Verification (CAV'97)*, volume 1254 of *Lecture Notes in Computer Science*, pages 352–363. Springer-Verlag, June 1997.

W. Marek and M. Truszczyński. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: a 25-Year Perspective*, pages 375–398. Springer-Verlag, 1999.

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

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

L. Robert Pokorny and C.R. Ramakrishnan. Model checking linear temporal logic using tabled logic programming. In *Workshop on Tabulation in Parsing and Deduction*, 2000.

C.R. Ramakrishnan, I.V. Ramakrishnan, S. A. Smolka, Yifei Dong, Xiaoqun Du, Abhik Roychoudhury, and V.N. Venkatakrishnan. XMC: A logic-programming-based verification toolset. In *Proceedings of the 12th International Conference on Computer Aided Verification (CAV'2000)*, volume 1855 of *Lecture Notes in Computer Science*, pages 576–580. Springer-Verlag, 2000.

K. Varpaaniemi, K. Heljanko, and J. Lilius. PROD 3.2 - An advanced tool for efficient reachability analysis. In *Proceedings of the 9th International Conference on Computer Aided Verification (CAV'97)*, volume 1254 of *Lecture Notes in Computer Science*, pages 472–475. Springer-Verlag, June 1997.

Appendix A Proofs

A.1 Proof of Theorem 2

We first recall our proof objective. Let $N = \langle P, T, F \rangle$ be a 1-safe P/T-net for all initial markings satisfying a condition C_0.

We want to prove that the 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.

The proof is based on the following two lemmata which establish a correspondence between stable models of $\Pi_{\mathrm{M}}(C_0, 0) \cup \Pi_{\mathrm{A}}(N, n)$ and n-bounded step executions of the 1-safe P/T-net N. We say that a step execution

$$\sigma_{N,n}(\Delta) = M_0 \overset{S_0}{\to} M_1 \overset{S_1}{\to} \ldots M_{n-1} \overset{S_{n-1}}{\to} M_n \qquad \text{(A1)}$$

is *derived from* a stable model Δ if for all $i = 0, \ldots, n$, $M_i = \{p \in P \mid p(i) \in \Delta\}$ and for all $i = 0, \ldots, n-1$, $S_i = \{t \in T \mid t(i) \in \Delta\}$.

Lemma 1
Let $N = \langle P, T, F \rangle$ be a 1-safe P/T-net for all initial markings satisfying a condition C_0. If $\Pi_{\mathrm{M}}(C_0, 0) \cup \Pi_{\mathrm{A}}(N, n)$ has a stable model Δ, then $\sigma_{N,n}(\Delta)$ is a step execution of N starting from an initial marking satisfying C_0.

Proof
Consider a step execution $\sigma_{N,n}(\Delta)$ (A1) which is derived from a stable model Δ of $\Pi_{\mathrm{M}}(C_0, 0) \cup \Pi_{\mathrm{A}}(N, n)$. Because Δ satisfies rules $\Pi_{\mathrm{M}}(C_0, 0)$, then marking M_0 satisfies condition C_0. Now we show that $\sigma_{N,n}(\Delta)$ is a valid step execution starting from M_0 by showing that if the step execution is valid up to marking M_i, then it is valid also up to M_{i+1}, i.e., $M_i \overset{S_i}{\rightarrow} M_{i+1}$ holds for all $i = 0, \ldots, n-1$. Consider $S_i = \{t \in T \mid t(i) \in \Delta\}$. As every stable model is supported, $t(i) \in \Delta$ implies that there is a rule in $\Pi_{\mathrm{M}}(C_0, 0) \cup \Pi_{\mathrm{A}}(N, n)$ with $t(i)$ as the head and the body literals satisfied in Δ. The only candidate rule is (5) and, hence, for every place p in the preset of t, $p(i) \in \Delta$ and, thus, $p \in M_i$. This implies that every transition $t \in S_i$ is enabled in M_i. Moreover, as Δ satisfies rules (7), S_i is concurrently enabled in M_i.

Given a marking M_i and a concurrently enabled step S_i, $M_i \overset{S_i}{\rightarrow} M_{i+1}$ holds in a 1-safe net, if for all $p \in P$, $p \in M_{i+1}$ iff

$$\text{(a) } p \in t^{\bullet} \text{ for some } t \in S_i \text{ or (b) } p \in M_i \text{ and for all } t \in S_i, p \notin {}^{\bullet}t. \qquad \text{(A2)}$$

We complete the proof by showing that this holds for M_{i+1}. Consider a place $p \in P$.

($\Rightarrow$) If $p \in M_{i+1}$, then $p(i+1) \in \Delta$. Hence, there is some rule in $\Pi_{\mathrm{A}}(N, n)$ with $p(i+1)$ as the head and the body literals satisfied in Δ. There are two types of candidate rules (6) and (8). In the case of (6), if the body is satisfied in Δ, $t(i) \in \Delta$ and $t \in S_i$ for a transition t with $p \in t^{\bullet}$ implying that Condition (A2: a) holds. For (8), if the body is satisfied in Δ, then $p \in M_i$ and no transition having p in its preset is in S_i. This implies that Condition (A2: b) holds.

($\Leftarrow$) If Condition (A2: a) holds for $p \in P$, then there is some $t(i) \in \Delta$. Because a rule $p(i+1) \leftarrow t(i)$ of type (6) is in $\Pi_{\mathrm{A}}(N, n)$, $p(i+1) \in \Delta$ and, hence, $p \in M_{i+1}$. If Condition (A2: b) holds for $p \in P$, then $p(i) \in \Delta$ and for all transition t with $p \notin {}^{\bullet}t$, $t(i) \notin \Delta$. As Δ satisfies a rule (8) for $p(i+1)$, $p(i+1) \in \Delta$ and, hence, $p \in M_{i+1}$. $\square$

Lemma 2
Let $N = \langle P, T, F \rangle$ be a 1-safe P/T-net for all initial markings satisfying a condition C_0. If there is a step execution σ' of N without empty steps from an initial marking M_0 satisfying C_0 containing $n' \leq n$ steps, then there is a stable model Δ of $\Pi_{\mathrm{M}}(C_0, 0) \cup \Pi_{\mathrm{A}}(N, n)$ such that the derived step execution $\sigma = \sigma_{N,n}(\Delta) = M_0 \overset{S_0}{\rightarrow} M_1 \overset{S_1}{\rightarrow} \ldots M_{n-1} \overset{S_{n-1}}{\rightarrow} M_n$ is a step execution of N, such that σ is the execution σ' with $n - n'$ empty steps added to the beginning.

Proof

Let σ' be a step execution from an initial marking M_0 satisfying C_0 in $n' \leq n$ steps. Then there is a step execution

$$M_0 \overset{S_0}{\to} M_1 \overset{S_1}{\to} \dots M_{n-1} \overset{S_{n-1}}{\to} M_n \tag{A3}$$

where $n - n'$ first steps are empty if $n' < n$, i.e., $S_0 = \cdots = S_{n-n'-1} = \{\}$ and $M_0 = \cdots = M_{n-n'}$.

Now consider a set of atoms

$$\begin{aligned}\Delta \;=\; & \{p(i) \mid p \in M_i, 0 \leq i \leq n\} \cup \\ & \{t(i) \mid t \in S_i, 0 \leq i < n\} \cup \\ & \{\mathit{idle}(0), \dots, \mathit{idle}(n-n'-1)\} \cup \\ & \{p'(0) \mid p \in P - M_0\} \cup \{t'(i) \mid t \in T - S_i, 0 \leq i < n\} \cup C(0)\end{aligned}$$

where $C(0)$ are the atoms $c_f(0)$ corresponding to the subexpressions f of Condition C_0 that are satisfied in M_0.

We show that Δ is a stable model of $\Pi = \Pi_{\mathrm{M}}(C_0, 0) \cup \Pi_{\mathrm{A}}(N, n)$ by establishing that (i) $\Delta \models \Pi^{\Delta}$ and that (ii) if $\Delta' \subseteq \Delta$ and $\Delta' \models \Pi^{\Delta}$, then $\Delta \subseteq \Delta'$ which together imply that Δ is the minimal set of atoms satisfying Π^{Δ}.

(i) By construction the rules in $\Pi_{\mathrm{M}}(C_0, 0)^{\Delta}$ are satisfied by Δ. Now we consider other rules in $\Pi_{\mathrm{A}}(N, n)$ case by case and show that rules resulting from them in Π^{Δ} are satisfied. Rules resulting from (4) and (5) are satisfied directly by construction of Δ because $p(0) \notin \Delta$ iff $p'(0) \in \Delta$ and $t(i) \notin \Delta$ iff $t'(i) \in \Delta$. Consider a rule (6) and assume that $t(i) \in \Delta$. Now $t \in S_i$ with $p \in t^{\bullet}$. This implies that $p \in M_{i+1}$ and, hence, $p(i+1) \in \Delta$. Each rule (7) is satisfied by Δ because each S_i is concurrently enabled implying that no S_i can contain any two transition sharing place in their presets. Consider the reduct $p(i+1) \leftarrow p(i) \in \Pi^{\Delta}$ of a rule (8) and the case where $p(i) \in \Delta$. Now $p \in M_i$ and for each transition t with p in its postset $t(i) \notin \Delta$. Hence, there is no transition with p in its preset in S_i implying that $p \in M_{i+1}$ and $p(i+1) \in \Delta$. Rules (9) are straightforwardly satisfied by construction of Δ. Hence, $\Delta \models \Pi^{\Delta}$ holds.

(ii) Consider a set $\Delta' \subseteq \Delta$ such that $\Delta' \models \Pi^{\Delta}$. Assume that there is an atom $x \in \Delta - \Delta'$. This atom cannot be any $p(0)$ for a place p because for each $p(0) \in \Delta$ there is a fact $p(0) \leftarrow \in \Pi^{\Delta}$. Similarly, it cannot be any $p'(0)$, $t'(i)$ for some $t \in T$ or $\mathit{idle}(i)$ because also for each of these there is a corresponding fact in Π^{Δ}.

Hence, x is either some $p(i)$ with $p \in P$ and $0 < i \leq n$ or some $t(i)$ with $t \in T$ and $0 \leq i \leq n$. Now consider such an atom x with the smallest index i. Suppose x is some $p(i) \in \Delta - \Delta'$. Then $p \in M_i$ which implies that (a) $p \in t^{\bullet}$ for some $t \in S_{i-1}$ or (b) $p \in M_{i-1}$ and for all $t \in S_{i-1}, p \notin {}^{\bullet}t$. In the case (a) there is some $t(i-1) \in \Delta'$ and as Δ' satisfies a rule of type (6) for $p(i)$, $p(i) \in \Delta'$. In the case (b), $p(i) \leftarrow p(i-1) \in \Pi^{\Delta}$ and $p(i-1) \in \Delta'$ which implies $p(i) \in \Delta'$. Hence, in both cases $p(i) \in \Delta'$ holds implying that x must be some $t(i)$ with $t \in T$ and $0 \leq i \leq n$. As $t(i) \in \Delta$, $t \in S_i$ implying that t is enabled and, hence, that every place p in the preset of t is in M_i. But then for every place p in the preset of t, $p(i) \in \Delta$

and, hence, $p(i) \in \Delta'$. As Δ' satisfies the rule $t(i) \leftarrow p_1(i), \ldots, p_l(i) \in \Pi^{\Delta}$ where $\{p_1, \ldots, p_l\}$ is the preset of t, $t(i) \in \Delta'$, a contradiction. Thus, $\Delta \subseteq \Delta'$. $\Box$

Proof
(of Theorem 2).

($\Leftarrow$) If $\Pi_{\mathrm{M}}(C_0, 0) \cup \Pi_{\mathrm{A}}(N, n) \cup \Pi_{\mathrm{M}}(C, n)$ has a stable model Δ, then by Proposition 1 there is a stable model $\Delta_E = \Delta \cap At$ of $\Pi_{\mathrm{M}}(C_0, 0) \cup \Pi_{\mathrm{A}}(N, n)$ where $At = \mathrm{Atoms}(\Pi_{\mathrm{M}}(C_0, 0) \cup \Pi_{\mathrm{A}}(N, n))$. By Lemma 1 $\sigma_{N,n}(\Delta_E)$ is a step execution of N starting from an initial marking satisfying C_0. As Δ satisfies rule $\Pi_{\mathrm{M}}(C, n)$, then the marking M_n in $\sigma_{N,n}(\Delta_E)$ satisfies condition C.

($\Rightarrow$) If N has an initial marking M_0 satisfying C_0 such that a marking M satisfying condition C is reachable in at most n steps, then by Lemma 2 $\Pi_{\mathrm{M}}(C_0, 0) \cup \Pi_{\mathrm{A}}(N, n)$ has a stable model Δ_E such that the derived step execution $\sigma_{N,n}(\Delta_E) = M_0 \overset{S_0}{\to} M_1 \overset{S_1}{\to} \ldots M_{n-1} \overset{S_{n-1}}{\to} M_n$ is a step execution of N and $M = M_n$. The rules $\Pi_{\mathrm{M}}(C, n) - \{\leftarrow \text{not } c(n)\}$ are stratified and the heads do not occur in $\Pi_{\mathrm{M}}(C_0, 0) \cup \Pi_{\mathrm{A}}(N, n)$. By Proposition 2 there is a unique stable model Δ of $\Pi_{\mathrm{M}}(C_0, 0) \cup \Pi_{\mathrm{A}}(N, n) \cup \Pi_{\mathrm{M}}(C, n) - \{\leftarrow \text{not } c(n)\}$ such that $\Delta_E = \Delta \cap At$ where $At = \mathrm{Atoms}(\Pi_{\mathrm{M}}(C_0, 0) \cup \Pi_{\mathrm{A}}(N, n))$. As M_n satisfies condition C, then $c(n) \in \Delta$ and, hence, $\leftarrow$ not $c(n)$ is satisfied by Δ implying by Proposition 3 that Δ is a stable model of $\Pi_{\mathrm{M}}(C_0, 0) \cup \Pi_{\mathrm{A}}(N, n) \cup \Pi_{\mathrm{M}}(C, n)$. This concludes our proof of Theorem 2. $\Box$

A.2 Proof of Theorem 3

We first recall our proof objective. Let f be an LTL formula in positive normal form and $N = \langle P, T, F \rangle$ be a 1-safe P/T-net for all initial markings satisfying a condition C_0.

We want to prove that whenever we have a stable model Δ_{LTL} of the program $\Pi = \Pi_{\mathrm{M}}(C_0, 0) \cup \Pi_{\mathrm{A}}(N, n) \cup \Pi_{\mathrm{L}}(N, n) \cup \Pi_{\mathrm{LTL}}(f, n)$ we can construct a maximal execution of the net system N from an initial marking satisfying C_0 which satisfies f.

Our proof proceeds as follows. We first derive a step execution σ' from the stable model Δ_{LTL}. We then create a maximal step execution σ'' from σ' using an index $0 \leq l \leq n$ also obtained from Δ_{LTL}. After this we show that a maximal (interleaving) execution σ can be obtained from σ'' such that $\sigma \models f$ iff $\sigma'' \models f$. Finally we show that $\sigma \models f$.

Lemma 3
For the stable model Δ_{LTL}, there is a step execution σ' of the net system N from an initial marking satisfying C_0.

Proof
We first use Proposition 1 with $\Pi_1 = \Pi_{\mathrm{M}}(C_0, 0) \cup \Pi_{\mathrm{A}}(N, n)$ and $\Pi_2 = \Pi_{\mathrm{L}}(N, n) \cup \Pi_{\mathrm{LTL}}(f, n)$ to obtain a stable model Δ_1 of the subprogram Π_1. By Lemma 1 the execution $\sigma' = \sigma_{N,n}(\Delta_1)$ is a step execution of N starting from an initial marking satisfying C_0. $\Box$

Let Δ_1 be the stable model and σ' the step execution obtained in the proof of Lemma 3 above. Now we show that by adding the rules $\Pi_{\mathrm{L}}(N,n)$ we can evaluate whether the last marking reached by σ' is a deadlock.

Lemma 4
Let Δ_1 be a stable model of $\Pi_1 = \Pi_{\mathrm{M}}(C_0,0) \cup \Pi_{\mathrm{A}}(N,n)$ and Δ_2 a stable model of $\Pi_2 = \Pi_1 \cup \Pi_{\mathrm{L}}(N,n)$. Then *live* $\in \Delta_2$ iff the last marking reached by $\sigma_{N,n}(\Delta_1)$ is *not* a deadlock.

Proof
The rules in $\Pi_{\mathrm{L}}(N,n)$ are stratified. Hence, by Proposition 2 Δ_2 is the unique stable model of Π_2 such that $\Delta_1 = \Delta_2 \cap \mathrm{Atoms}(\Pi_1)$. If *live* $\in \Delta_2$, then there is some rule in $\Pi_{\mathrm{L}}(N,n)$ with its body satisfied by Δ_2. As $\Delta_1 = \Delta_2 \cap \mathrm{Atoms}(\Pi_1)$, the body is satisfied by Δ_1 and, hence, there is an enabled transition in the last marking reached by $\sigma_{N,n}(\Delta_1)$. In the other direction, if there is an enabled transition t in the last marking, then $\{p_1(n),\ldots,p_l(n)\} \subseteq \Delta_1$ where $\{p_1,\ldots,p_l\}$ is the preset of t. But then $\{p_1(n),\ldots,p_l(n)\} \subseteq \Delta_2$ and *live* $\in \Delta_2$. $\square$

Hence, we can again use Proposition 1 with $\Pi_1 = \Pi_{\mathrm{M}}(C_0,0) \cup \Pi_{\mathrm{A}}(N,n) \cup \Pi_{\mathrm{L}}(N,n)$ and $\Pi_2 = \Pi_{\mathrm{LTL}}(f,n)$ together with Lemma 4 to show that *live* $\in \Delta_{LTL}$ iff the last marking reached by σ' is *not* a deadlock.

We now do a case analysis on three different types of counterexamples. The stable model Δ_{LTL} belongs to exactly one of the following three mutually exclusive cases:

a) $nl(l+1) \in \Delta_{LTL}$ for some $0 \le l \le n-1$: infinite maximal execution which we will represent as a pair (σ', l), where $0 \le l \le n-1$ such that $nl(l+1) \in \Delta_{LTL}$,
b) $nl(i+1) \notin \Delta_{LTL}$ for all $0 \le i \le n-1$, *live* $\notin \Delta_{LTL}$: finite maximal execution which we will represent as a pair (σ', n), or
c) $nl(i+1) \notin \Delta_{LTL}$ for all $0 \le i \le n-1$, *live* $\in \Delta_{LTL}$: non-maximal execution which we will also represent as a pair (σ', n).

We will now analyse the stable model Δ_{LTL}.

Lemma 5
The following holds for the three different cases of Δ_{LTL}.

a) If $nl(l+1) \in \Delta_{LTL}$, then $nl(i) \notin \Delta_{LTL}$ for all $i \neq (l+1)$, $le \in \Delta_{LTL}$, $il(i) \in \Delta_{LTL}$ for all $l+1 \le i \le n$ and *live* $\in \Delta_{LTL}$.
b) If $nl(i+1) \notin \Delta_{LTL}$ for all $0 \le i \le n-1$ and *live* $\notin \Delta_{LTL}$, then $le \notin \Delta_{LTL}$, and *live* $\notin \Delta_{LTL}$.
c) If $nl(i+1) \notin \Delta_{LTL}$ for all $0 \le i \le n-1$, and *live* $\in \Delta_{LTL}$, then $le \notin \Delta_{LTL}$.

Proof
a) Because the only rule with $nl(l+1)$ as head is $nl(l+1) \leftarrow el(l)$ we also get that $el(l) \in \Delta_{LTL}$. Because rule (12) is satisfied we know that for all $i \neq l$ it holds that $el(i) \notin \Delta_{LTL}$. Because $le \leftarrow el(l) \in \Pi$, we also know that $le \in \Delta_{LTL}$. By using the rules $il(i+1) \leftarrow el(i)$ and $il(i+1) \leftarrow il(i)$ and the fact that $el(l) \in \Delta_{LTL}$ combined with simple induction we get that that $il(i) \in \Delta_{LTL}$ for all $l+1 \le i \le n$. The rules

(13) imply that $p \in M_l$ iff $p \in M_n$. From the rule (15) and $le \in \Delta_{LTL}$ we get that $idle(n-1) \notin \Delta_{LTL}$ and thus the step S_{n-1} is non-empty. Taking together that $M_l = M_n$ and that the step S_{n-1} is non-empty implies M_n is not a deadlock, and thus $live \in \Delta_{LTL}$.

b,c) Because $nl(i+1) \notin \Delta_{LTL}$ for all $0 \leq i \leq n-1$, we know also that $el(i) \notin \Delta_{LTL}$ for all $0 \leq i \leq n-1$. Then as the only rules having le as head are the rules (14) of the form $le \leftarrow el(i)$, $le \notin \Delta_{LTL}$.

□

We record some facts discovered in the proof of Lemma 5, case a), in the following.

Corollary 2
In the case a) for σ' it holds that $M_l = M_n$ and the step S_{n-1} is non-empty.

We will next state an additional property of σ' and show how a maximal step execution σ'' can be obtained given the pair (σ', l).

Lemma 6
Each step of σ' contains at most one visible transition.

Proof
We use Proposition 3 with the rules (16) of the subprogram $\Pi_{\mathrm{LTL}}(f, n)$. □

Lemma 7
For the stable model Δ_{LTL} there is a maximal step execution σ'' of the net system N from an initial marking satisfying C_0.

Proof
In all cases below M_0 is the initial marking of σ' and thus satisfies C_0.

In the case a) we know from Corollary 2 that $M_l = M_n$ and we can thus generate an infinite maximal step execution of N using the pair (σ', l). The corresponding infinite step execution σ'' is

$$M_0 \xrightarrow{S_0} M_1 \xrightarrow{S_1} \cdots M_{n-1} \xrightarrow{S_{n-1}} M_n \xrightarrow{S_l} M_{l+1} \xrightarrow{S_{l+1}} \cdots M_{n-1} \xrightarrow{S_{n-1}} M_n \xrightarrow{S_l} \cdots$$

We also know from Corollary 2 that the step S_{n-1} is non-empty. Therefore σ'' contains infinitely many non-empty steps.

In the case b) the step execution $\sigma'' = \sigma'$ will be a maximal step execution of N.

In the case c) we can pick an *interleaving* execution σ''' such that the concatenation of σ' followed by σ''' will be a maximal step execution σ'' of N. □

We can now state the existence of maximal executions given the stable model Δ_{LTL}.

Lemma 8
For the stable model Δ_{LTL} there is a maximal (interleaving) execution σ of the net system N from an initial marking satisfying C_0.

Proof
By using the procedure described above we can obtain the maximal step execution σ'' from Δ_{LTL}.

By removing all idle time steps from the maximal step execution σ'' of N, and replacing each step by its linearisation, i.e, by some permutation of transitions that make up the step, we can construct a maximal *interleaving* execution σ of N. The initial marking M_0 of σ'' is also the initial marking of σ and thus satisfies C_0. $\square$

Let $w, w', w'' \in V^+ \cup V^\omega$ be the words corresponding to the step executions $\sigma, \sigma', \sigma''$ discussed above, respectively. What we prove next is that $w \models f$ iff $w'' \models f$ for the LTL formula f.

We need a technical notion of stuttering equivalence for words. The intuition behind this equivalence is that if two words are stuttering equivalent, they satisfy exactly the same LTL formulas.[3] Our definition of stuttering equivalence is motivated by a similar definition in Chapter 10.2 of (CGP99) where also a longer discussion of its use can be found.

Definition 3
Two words $v, v' \in V^+ \cup V^\omega$ are stuttering equivalent when:

- Both are infinite words $v, v' \in V^\omega$, and there two infinite sequences of positive integers $0 = i_0 < i_1 < i_2 < \ldots$ and $0 = j_0 < j_1 < j_2 < \ldots$ such that for every $k \geq 0 : v_{(i_k)} = v_{(i_k+1)} = \ldots = v_{(i_{(k+1)}-1)} = v'_{(j_k)} = v'_{(j_k+1)} = \ldots = v'_{(j_{(k+1)}-1)}$, or
- both are finite words $v, v' \in V^+$, and there exist an integer $n \geq 1$ and two finite sequences of positive integers $0 = i_0 < i_1 < \ldots < i_n = |v|$ and $0 = j_0 < j_1 < \ldots < j_n = |v'|$ such that for every $0 \leq k < n : v_{(i_k)} = v_{(i_k+1)} = \ldots = v_{(i_{(k+1)}-1)} = v'_{(j_k)} = v'_{(j_k+1)} = \ldots = v'_{(j_{(k+1)}-1)}$.

The following proposition can be proved using a simple induction on the structure of the formula f using the definition of LTL semantics.

Proposition 4
Let f be an LTL formula and v, v' be two stuttering equivalent words. Then $v \models f$ iff $v' \models f$.

Lemma 9
The words w and w'' corresponding to the maximal execution σ and the maximal step execution σ'' are stuttering equivalent.

[3] This property crucially depends on the non-existence of the next-time operator $X\varphi_1$ in our definition of LTL.

Proof
Lemma 6 implies that each step in σ' consists of at most one visible transition, and in case c) the suffix σ''' is interleaving by definition. Thus each step of the execution σ'' contains at most one visible transition. Thus when a step is replaced by some linearisation in the proof of Lemma 8, the step is changed to (possibly empty) stuttering of the original atomic propositions, (possibly) followed by change in them, followed by (possibly empty) stuttering of the new atomic propositions. Thus each step is replaced by a stuttering equivalent sequence, implying that the whole sequence is stuttering equivalent. $\square$

Lemma 9 implies that if we correctly evaluate the LTL formula f for the word w'' then we also correctly evaluate it for w. All we have to know to correctly evaluate the LTL formula for the word w'' is to know the prefix word w', the index l, and whether we are in case a), b), or c). The evaluation for case a) and b) will be exact, while the case c) will only be approximate, as in that case nothing is known about the suffix of the word w'.

Evaluating the formula f. Assume we are given a finite word $u \in V^+$ of length $n+1$, index $0 \leq l \leq n$, and knowledge whether we are in case a), b), or c). This induces a word $y \in V^+ \cup V^\omega$ such that in the case a) $y = u(u^{(l+1)})^\omega$ and in the cases b) and c) $y = u$.

Given sufficient assumptions about a base program, call it Π_B, and its stable model Δ_B, we want to show that by adding to it the translation of formula f as given by Fig. 4 we obtain a program Π_C, whose unique stable model Δ_C respects the semantics of LTL in the following sense for all three different cases and $0 \leq i \leq n$:

a) $f(i) \in \Delta_C$ iff $y^{(i)} \models f$ for $y \in V^\omega$,
b) $f(i) \in \Delta_C$ iff $y^{(i)} \models f$ for $y \in V^+$, and
c) if $f(i) \in \Delta_C$ then $y' \models f$ for all $y'' \in V^+ \cup V^\omega$ such that $y' = y^{(i)}y''$.

The case c) specifies only a prefix y of a word. Our encoding cautiously under-approximates the semantics of LTL formulas in the presence of uncertainty about the suffix y''.

Notice also that in the case a) the word y is cyclic, and the semantics of LTL follows the same cycle when $i \geq l$. Thus to evaluate, e.g., $y^{(n+1)} \models f$ it suffices to evaluate $y^{(l+1)} \models f$.

The assumptions on the program Π_B and its stable model Δ_B are as follows:

1. The atoms appearing as heads in the LTL translation do not occur in the program Π_B.
2. For all $p \in P$, $0 \leq i \leq n$: $p(i) \in \Delta_B$ iff $p(i) \in y_{(i)}$.
3. Δ_B is exactly one of the following three cases:

   a) $nl(l+1) \in \Delta_B$, $nl(i) \notin \Delta_B$ for all $i \neq (l+1)$, $le \in \Delta_B$, $il(i) \in \Delta_B$ for all $l+1 \leq i \leq n$, and $live \in \Delta_B$.
   b) $nl(i+1) \notin \Delta_B$ for all $0 \leq i \leq n-1$, $le \notin \Delta_B$, and $live \notin \Delta_B$.
   c) $nl(i+1) \notin \Delta_B$ for all $0 \leq i \leq n-1$, $le \notin \Delta_B$, and $live \in \Delta_B$.

Lemma 10
If the assumptions stated above hold for a base program Π_B and its stable model Δ_B, then a program Π_C obtained from Π_B by adding the translation of LTL formulas as given by Fig. 4, has a stable model Δ_C, which follows the semantics of LTL for all $0 \leq i \leq n$.

Proof
First we note that Assumption 1. above together with Proposition 2 and the fact that the translation as given by Fig. 4 is stratified imply that a stable model Δ_C of the combined program exists, and is unique.

We now do the proof by induction on the structure of the formula f. Assume that the translation of the subformulas f_1 and f_2 follow the semantics of LTL. Then we prove that also the translation for f follows the semantics of LTL.

We do a case split by the formula type:

- $f = p$, for $p \in AP$, or $f = \neg p$, for $p \in AP$:
By Assumption 2. above $p(i) \in \Delta_C$ iff $p(i) \in y_{(i)}$.
- $f = f_1 \vee f_2$, or $f = f_1 \wedge f_2$:
The translation directly follows the semantics of LTL.
- $f = f_1 \, U \, f_2$:
We show that f follows the semantics of LTL for all $0 \leq i \leq n$ by establishing that (i) it does that for $i = n$ and that (ii) if f follows the semantics of LTL for $i + 1$, then it does for i. The proof is based on the following equivalence valid for the U operator for all $0 \leq i < n$:

$$y^{(i)} \models f_1 \, U \, f_2 \text{ iff } y^{(i)} \models f_2 \text{ or } (y^{(i)} \models f_1 \text{ and } y^{(i+1)} \models f_1 \, U \, f_2) \qquad \text{(A4)}$$

(i) Suppose $f(n) \in \Delta_C$. In the cases b) and c) $f(n + 1) \notin \Delta_C$, implying that $f_2(n) \in \Delta_C$ because $f(n) \leftarrow f_2(n)$ is the only rule supporting $f(n)$. Hence, $y^{(n)} \models f$ holds and $y' \models f$ for any y' extending $y^{(n)}$ in the case c). Consider now the case a). Suppose there is no $f_2(j) \in \Delta_C$ with $l < j \leq n$. Then for $\Delta'_C = \Delta_C - \{f(j) \mid l < j \leq n\}$, $\Delta'_C \models \Pi_C^{\Delta_C}$ and $\Delta'_C \subset \Delta_C$ which implies that Δ_C is not a stable model of Π_C, a contradiction. Hence, there is some $f_2(j) \in \Delta_C$ with $l < j \leq n$. Take such $f_2(j) \in \Delta_C$ with the smallest index j. Suppose there is some $f_1(j') \notin \Delta_C$ with $l < j' < j$. Now for $\Delta'_C = \Delta_C - \{f(j')\}$, $\Delta'_C \models \Pi_C^{\Delta_C}$ and $\Delta'_C \subset \Delta_C$ implying that Δ_C is not a stable model of Π_C, a contradiction. Hence, for all $l < j' < j$, $f_1(j) \in \Delta_C$. This implies by the inductive hypothesis that $y^{(n)} \models f$ holds.
Suppose $y^{(n)} \models f$ holds. Then in the case b) $y^{(n)} \models f_2$ holds and hence, by rule $f(n) \leftarrow f_2(n) \in \Pi_C$, $f(n) \in \Delta_C$. In the case a) there is some $f_2(j) \in \Delta_C$ with $l < j \leq n$, and for all $l < j' \leq j$, $f_1(j') \in \Delta_C$. Then by rules in the translation of f, $f(n) \in \Delta_C$.
(ii) Suppose $f(i) \in \Delta_C$, $i < n$. Then there is a supporting rule in $\Pi_C^{\Delta_C}$ with $f(i)$ in the head and the body literals satisfied in Δ_C. There are two candidate rules $f(i) \leftarrow f_2(i)$ and $f(i) \leftarrow f_1(i), f(i + 1)$. By the inductive hypotheses in the first case $y^{(i)} \models f_2$ and in the second case $y^{(i)} \models f_1$ and $y^{(i+1)} \models f$ which imply by (A4) $y^{(i)} \models f$. In the other direction for cases a) and b), if $y^{(i)} \models f$, then by (A4)

$y^{(i)} \models f_2$ or ($y^{(i)} \models f_1$ and $y^{(i+1)} \models f$). From these using the rules $f(i) \leftarrow f_2(i)$ and $f(i) \leftarrow f_1(i), f(i+1)$ in Δ_C and the inductive hypotheses follows that $f(i) \in \Delta_C$ holds.

- $f = f_1 \, R \, f_2$:
  We show that f follows the semantics of LTL for all $0 \le i \le n$ by establishing that (i) it does that for $i = n$ and that (ii) if f follows the semantics of LTL for $i+1$, then it does for i.
  (i) Consider first the case b). Now $f(n+1) \notin \Delta_C$. If $f(n) \in \Delta_C$, then $f(n) \leftarrow f_2(n)$ and $f(n) \leftarrow f_2(n), f_1(n)$ are the only rules supporting $f(n)$ in $\Pi_C^{\Delta_C}$. Hence, $f_2(n) \in \Delta_C$, $y^{(n)} \models f_2$ and thus $y^{(n)} \models f$. In the other direction, if $y^{(n)} \models f$, then $y^{(n)} \models f_2$ implying $f(n) \in \Delta_C$. In the case c) if $f(n) \in \Delta_C$, $f(n) \leftarrow f_2(n), f_1(n)$ is the only rule supporting $f(n)$ in $\Pi_C^{\Delta_C}$ and hence $y^{(n)} \models f_1 \wedge f_2$ which implies $y' \models f$ for any y' extending $y^{(n)}$. Thus for the cases b) and c) condition (i) holds.
  Now consider the case a) where we use the following equivalence between LTL formulas:
  $$y \models f_1 \, R \, f_2 \text{ iff } y \models (f_2 \, U \, (f_1 \wedge f_2)) \vee (\Box f_2). \tag{A5}$$
  We consider two cases

  — $y^{(n)} \models \Box f_2$
  In this case by (A5) $y^{(n)} \models f$ but also $f_2(j) \in \Delta_C$ for all $l < j \le n$ which implies that $c(f) \notin \Delta_C$ and $f(n+1) \in \Delta_C$ and hence $f(n) \in \Delta_C$.

  — $y^{(n)} \not\models \Box f_2$
  In this case by (A5) $y^{(n)} \models f$ iff $y^{(n)} \models (f_2 \, U \, (f_1 \wedge f_2))$. Now we can show that $y^{(n)} \models f$ iff $f(n) \in \Delta_C$ using a similar argument as in the previous case for the U operator. This is because $y^{(n)} \not\models \Box f_2$ implies that there is some $f_2(j) \notin \Delta_C$ with $l < j \le n$. Hence, $c(f) \in \Delta_C$. Then the rules for $f(i)$ in $\Pi_C^{\Delta_C}$ are
  $$\begin{aligned} &f(i) \leftarrow f_2(i), f_1(i) \\ &f(i) \leftarrow f_2(i), f(i+1) \\ &f(n+1) \leftarrow nl(i), f(i) \end{aligned}$$
  which would be the evaluation rules for the U formula $(f_2 \, U \, (f_1 \wedge f_2))$.

  Hence, in both case $y^{(n)} \models f$ iff $f(n) \in \Delta_C$ and thus for the case a) condition (i) holds.
  (ii) We use the following equivalence valid for all $0 \le i < n$:
  $$y^{(i)} \models f_1 \, R \, f_2 \text{ iff } y^{(i)} \models f_2 \wedge f_1 \text{ or } (y^{(i)} \models f_2 \text{ and } y^{(i+1)} \models f_1 \, R \, f_2) \tag{A6}$$
  If $y^{(i)} \models f$, then by (A6) $f_2(i), f_1(i) \in \Delta_C$ or $f_2(i), f(i+1) \in \Delta_C$, which imply $f(i) \in \Delta_C$. In the other direction, if $f(i) \in \Delta_C$, then there are two possible rules in $\Pi_C^{\Delta_C}$ supporting $f(i)$: $f(i) \leftarrow f_2(i), f_1(i)$ and $f(i) \leftarrow f_2(i), f(i+1)$. Hence, $f_2(i), f_1(i) \in \Delta_C$ or $f_2(i), f(i+1) \in \Delta_C$, which imply by (A6) that $y^{(i)} \models f$. Hence, condition (ii) holds.

□

Final proof of Theorem 3. Let Δ_{LTL} be a stable of the program $\Pi = \Pi_{\mathrm{M}}(C_0, 0) \cup \Pi_{\mathrm{A}}(N, n) \cup \Pi_{\mathrm{L}}(N, n) \cup \Pi_{\mathrm{LTL}}(f, n)$. By Lemma 8 we can obtain from Δ_{LTL} a maximal (interleaving) execution σ of N from an initial marking satisfying C_0. Consider now the subprogram Π_B, which consist of $\Pi_{\mathrm{M}}(C_0, 0) \cup \Pi_{\mathrm{A}}(N, n) \cup \Pi_{\mathrm{L}}(N, n)$ and the rules (11)-(16) of $\Pi_{\mathrm{LTL}}(f, n)$. By Proposition 1 and Lemma 5 the stable model Δ_{LTL} projected on the atoms of Π_B satisfies the assumptions required by Lemma 10. Now Lemma 7 and Lemma 10 with $u = w'$ imply that if $f(0)$ is in a stable model Δ_{LTL}, then for the word w'' corresponding to the maximal step execution σ'' it holds that $w'' \models f$. Because the word w'' is stuttering equivalent to the word w corresponding to σ according to Lemma 9, Proposition 4 implies that if $f(0)$ is in a stable model Δ_{LTL}, then $\sigma \models f$. The rule (17) implies that $f(0) \in \Delta_{LTL}$, and thus $\sigma \models f$. This completes our proof of Theorem 3. □

A.3 Proof of Theorem 4

We first recall our proof objective. Let f be an LTL formula in positive normal form and $N = \langle P, T, F \rangle$ be a 1-safe P/T-net for all initial markings satisfying a condition C_0.

We want to prove that if N has a looping or deadlock execution of at most length n starting from an initial marking satisfying C_0 such that some corresponding maximal execution σ satisfies f, then $\Pi = \Pi_{\mathrm{M}}(C_0, 0) \cup \Pi_{\mathrm{A}}(N, n) \cup \Pi_{\mathrm{L}}(N, n) \cup \Pi_{\mathrm{LTL}}(f, n)$ has a stable model.

We thus know that there is a deadlock or looping execution, call it σ', of N of length n' such that $n' \leq n$ and for some corresponding maximal execution σ it holds that $\sigma \models f$.

There are now two mutually exclusive cases:

a) σ' is a looping execution. Notice that in this case $S_{n'-1} = \{t_{n'-1}\}$, i.e., the last step is always non-empty. Without loss of generality we select the minimal index $0 \leq l \leq n-1$ such that $M_l = M_n$ and such that the corresponding maximal execution $\sigma \models f$, where σ is the maximal execution which visits the sequence of states $M_0, M_1, \ldots, M_l, M_{l+1}, \ldots, M_k, M_{l+1}, \ldots, M_k, \ldots$.
b) σ' is a deadlock execution. In this case $\sigma = \sigma'$ is a maximal execution such that $\sigma \models f$. We now set $l = n$ to differentiate from the previous case.

Note that the case c) used in proof of Theorem 3 is not needed here, as we do not consider non-maximal executions.

Lemma 2 implies that the program $\Pi_0 = \Pi_{\mathrm{M}}(C_0, 0) \cup \Pi_{\mathrm{A}}(N, n)$ has a stable model Δ_0 whose derived execution can be obtained from σ' by adding $n - n'$ empty steps at the beginning of the execution.

We keep the name σ' for the step execution derived by adding $n - n'$ idle steps to the beginning of σ', and add $n - n'$ idle steps to the beginning of the corresponding maximal execution σ. Also the loop point l is increased by $n - n'$ to compensate for the addition of idle steps. Clearly the obtained step execution σ' is of length n, is a deadlock execution iff the original is, and is a looping execution iff the original is. Moreover, the corresponding maximal execution σ satisfies the formula f also after the addition of the idle steps, as the word w corresponding to σ has $n - n'$ extra copies of stuttering of the initial atomic propositions to it, which by Proposition 4 cannot be detected by any LTL formula. Therefore we can from now on assume that σ' is of length (exactly) n.

Now consider the program Π_2 which consists of $\Pi_{\mathrm{M}}(C_0, 0) \cup \Pi_{\mathrm{A}}(N, n) \cup \Pi_{\mathrm{L}}(N, n)$ together with rules (11)-(16) of program $\Pi_{\mathrm{LTL}}(f, n)$. Given the pair (σ', l) we will show that the program Π_2 has a stable model Δ_2 capturing the essential properties of (σ', l).

Lemma 11
Given the pair (σ', l) we can construct a stable model Δ_2 of the program Π_2 such that the two claims stated below hold for Δ_2.

1. For all $p \in P$, $0 \leq i \leq n$: $p(i) \in \Delta_2$ iff $p(i) \in M_i$ in σ'.
2. Δ_2 is exactly one of the following two cases:
   a) If $0 \leq l \leq n-1$, then $nl(l+1) \in \Delta_2$, $nl(i) \notin \Delta_2$ for all $i \neq (l+1)$, $le \in \Delta_2$, $il(i) \in \Delta_2$ for all $l+1 \leq i \leq n$, and $live \in \Delta_2$.
   b) If $l = n$, then $nl(i+1) \notin \Delta_2$ for all $0 \leq i \leq n-1$, $le \notin \Delta_2$, and $live \notin \Delta_2$.

Proof
In the proofs below, use case a) when $0 \leq l \leq n-1$, and the case b) when $l = n$.

We know that the first claim holds for the stable model Δ_0 of the program Π_0 as σ' has been derived from it.

We also know from Lemma 4, and the fact that σ' has been derived from the stable model Δ_0 of Π_0 that the program $\Pi_1 = \Pi_0 \cup \Pi_{\mathrm{L}}(N, n)$ has a stable model Δ_1, such that

a) $\Delta_1 = \Delta_0 \cup \{live\}$, as a loop execution is not deadlocked after σ', or
b) $\Delta_1 = \Delta_0$, as σ' is a deadlock execution.

Given Π_1 and the stable model Δ_1, we will incrementally add rules to the program Π_1 whose head atoms do not appear in the program they are added into. At each step we prove that a stable model of the extended program exists. At the end we use Proposition 1 to project the final stable model Δ_2 on the atoms of Π_1 obtaining Δ_1, which fulfils the first claim and part of the second claim.

The rest of the second claim is proved incrementally by stating properties the stable models extending Δ_1 will have, finally ending up with the stable model Δ_2 of Π_2 which satisfies also the rest of the second claim.

1. First add the shorthand rules (11) to Π_1 obtaining the program Π_a. We show that a stable model Δ_a exists, which extends Δ_1 as follows. We do case analysis:
   a) $\Delta_a = \Delta_1 \cup \{el(l)\} \cup \{el(i)' \mid 0 \leq i \leq n-1 \text{ such that } i \neq l\}$
   b) $\Delta_a = \Delta_1 \cup \{el(i)' \mid 0 \leq i \leq n-1\}$

   In the case a) the shorthands (11) contribute to the reduct $\Pi_a^{\Delta_a}$ a fact $el(l) \leftarrow$ and a fact $el(i)' \leftarrow$ for every $0 \leq i \leq n-1$ such that $i \neq l$. Clearly, $\Delta_a \models \Pi_a^{\Delta_a}$ and, moreover, Δ_a is the smallest such set $\Delta \subseteq \Delta_a$ because removing $el(l)$ or one of

$el(i)'$ would leave the corresponding fact unsatisfied. Hence, Δ_a is a stable model of Π_a. The case b) is similar except that there is no fact $el(l) \leftarrow$ in the reduct but a fact $el(i)' \leftarrow$ for every $0 \leq i \leq n-1$.

2. Add the integrity constraints (12)-(13) to Π_a obtaining Π_b. Now $\Delta_b = \Delta_a$ is a stable model of Π_b. In the case a) the integrity constraint (12) is satisfied because $el(i) \in \Delta_b$ only for the index $i = l$. Because σ' is a loop execution with $M_l = M_k$ also all of the integrity constraints (13) are satisfied. In the case b) $el(i) \notin \Delta_b$ for all indices $0 \leq i \leq n-1$, and thus the integrity constraint (12) is satisfied and also all of the integrity constraints (13) are satisfied.
3. Add rules (14) to Π_b obtaining Π_c. The added rules are stratified, and thus by Proposition 2 a unique stable model Δ_c exists, which extends Δ_b as follows. We do a case analysis:

   a) $\Delta_c = \Delta_b \cup \{le, nl(l+1)\} \cup \{il(i) \mid l+1 \leq i \leq n\}$
   b) $\Delta_c = \Delta_b$

   The proof in the case a) proceeds starting from the fact that $el(l) \in \Delta_b$, from which we get $\{le, nl(l+1), il(l+1)\} \in \Delta_c$. We then get that $\{il(i) \mid l+2 \leq i \leq n\} \in \Delta_c$ by simple induction using the rules $il(i+1) \leftarrow il(i)$. In the case b) $el(i) \notin \Delta_b$ for all indices $0 \leq i \leq n-1$ implies that the rules (14) are satisfied by Δ_c, and thus Δ_c is a stable model.
4. Add the rule (15) to Π_c obtaining Π_d. This is an integrity constraint, which is satisfied in case a), as the step S_{n-1} is non-empty in looping executions. The integrity constraint is also satisfied in case b), as $le \notin \Delta_c$. Thus $\Delta_d = \Delta_c$ is in both cases a stable model of Π_d.
5. Finally, add the integrity constraints (16) to Π_d obtaining Π_e. These integrity constraints are always satisfied, as no time step in σ' contains more than one transition. Thus $\Delta_e = \Delta_d$ will be a stable model of Π_e.

By setting $\Delta_2 = \Delta_e$ we have shown that Δ_2 is a stable model of the program $\Pi_2 = \Pi_e$ such that Δ_2 projected on the atoms of Π_1 by Proposition 1 is Δ_1. This satisfies the first claim, and the part of the second claim concerning the atom *live*. The rest of the second claim has been incrementally proved in the cases above. □

Let Π_3 be the program which consists of Π_2 together with the translation of the LTL formula f as given by Fig. 4, and let w' be the finite word corresponding to the execution σ'.

Lemma 11 and Lemma 10 with $u = w'$ and $\Pi_B = \Pi_2$, and the fact that $\sigma \models f$ implies that the program Π_3 has a stable model such that $f(0) \in \Delta_3$. (Recall that we do not use case c), and thus our evaluation of f on the corresponding maximal execution σ is exact.) We can now add the constraint rule (17) to the program Π_3, and to obtain the full program Π. Now because $f(0) \in \Delta_3$ the integrity constraint (17) is satisfied, and we have found a stable model $\Delta = \Delta_3$ of Π. This completes our proof of Theorem 4. □