Model Checking LTL using Constraint Programming*

Javier Esparza and Stephan Melzer

Institut für Informatik
Arcisstraße 21
Technische Universität München
D-80333 München, Germany
e-mail: {esparza,melzers}@informatik.tu-muenchen.de

Abstract. The model-checking problem for 1-safe Petri nets and linear-time temporal logic (LTL) consists of deciding, given a 1-safe Petri net and a formula of LTL, whether the Petri net satisfies the property encoded by the formula. This paper introduces a semidecision test for this problem. By a semidecision test we understand a procedure which may answer 'yes', in which case the Petri net satisfies the property, or 'don't know'. The test is based on a variant of the so called *automata-theoretic approach* to model-checking and on the notion of T-invariant. We analyse the computational complexity of the test, implement it using *2lp* – a constraint programming tool, and apply it to two case studies.

Table of Contents

1 Introduction 3

2 The automata-theoretic approach to model-checking 4
- 2.1 Transition systems 4
- 2.2 Linear-time Temporal Logic 4
- 2.3 LTL on transition systems 5
- 2.4 Büchi Automata 5
- 2.5 Product automata 5
  - Action-based semantics 6
  - State-based semantics 6

3 Lifting the automata-theoretic model-checking method to Petri nets 6
- 3.1 Multiset Notation 6
- 3.2 Petri nets 7
- 3.3 LTL on 1-safe Petri nets 7
- 3.4 Büchi Nets 8
- 3.5 Product nets in action-based semantics 8
- 3.6 Product nets in state-based semantics 9

4 Testing emptiness of Büchi nets using T-invariants 13
- 4.1 Removing dead transitions 16

* This work is supported by the Sonderforschungsbereich SFB-342 A3. A (very) abbreviated version appears in the proceedings of Application and Theory of Petri Nets '97.

4.2 Combining Definition 5 and Definition 7 . . . 17
4.3 An improved Test . . . 18
4.4 Computational complexity . . . 20

5 An Implementation of the T*-Invariant Test Using Constraint Programming . . . 22

6 Applications . . . 26
6.1 A ring election algorithm . . . 26
Specification . . . 26
Implementation . . . 26
Verification and results . . . 27
6.2 A snapshot algorithm . . . 27
Specification . . . 28
Implementation . . . 28
Verification and results . . . 30

7 Simple LTL . . . 31

8 Conclusion . . . 35

A The M-net model . . . 36
A.1 Values, Variables and Bindings . . . 36
A.2 Actions and Action Terms . . . 36
A.3 M-nets, their Markings and the Firing Rule . . . 37
A.4 Some remarks on the M-net algebra . . . 38

B M-net Semantics of Büchi automata . . . 38
B.1 Preliminaries . . . 39
B.2 The Interfaces . . . 39
B.3 The Transformations . . . 39

1 Introduction

Linear-time temporal logic (LTL) [15] is a well-known formalism for specifying properties of concurrent systems. The problem of deciding if a concurrent system satisfies a LTL formula is called the *model-checking problem* (of LTL). In [34, 33] Vardi and Wolper introduced an *automata-theoretic approach* to this problem. The approach assumes that there exists a semantic mapping which associates to a concurrent system *sys* a *finite* (labelled) transition system A_{sys}. It asks the verifier to perform the following three tasks [18, 34]:

- Build a Büchi automaton $A_{\neg\phi}$ for the negation of the formula ϕ to be checked. $A_{\neg\phi}$ accepts exactly all infinite sequences that violate the formula ϕ.
- Construct a Büchi automaton A_p, called the *product* of A_{sys} and $A_{\neg\phi}$. A_p accepts all the infinite computations of A_{sys} that are accepted by $A_{\neg\phi}$, i.e., all infinite computations of A_{sys} that violate ϕ.
- Check whether the product automaton A_p is empty, i.e., whether it accepts no infinite sequences. A_{sys} satisfies ϕ iff A_p is empty.

The main problem of this approach is the well-known state-explosion phenomenon: the size of the transition system A_{sys} can grow exponentially in the size of *sys*. Several suggestions have been made to solve or at least palliate this problem: the transition system A_{sys} can be replaced by a trace automaton [18], and the size of A_{sys} can be reduced by means of different techniques like stubborn sets [31], sleep sets [18], or others.

In this paper we introduce still another technique to avoid the state-explosion, which can be applied when the system is modelled as a 1-safe Petri net. The technique is a *semidecision test*, that is, a procedure which may answer 'yes', in which case the property to be checked holds, or 'don't know'. A semidecision test has interest only if for relevant case studies it answers 'yes' and performs faster than exact methods. We provide evidence in this direction in the form of a complexity analysis and two case studies.

For systems modelled as Petri nets the transition system A_{sys} is just the well-known reachability graph. An straightforward application of the automata-theoretic approach would proceed by (1) building the reachability graph, and by (2) constructing the product automaton; it would obviously suffer from the state explosion problem. The first (minor) contribution of this paper is to show that step (2) can be performed before step (1). More specifically, we describe several ways of constructing a 'product Büchi net' N_p from a Petri net N_{sys} and a Büchi automaton $A_{\neg\phi}$. Using this construction it is immediate to reduce the model-checking problem to a certain 'net emptiness' problem, very similar to the emptiness problem of Büchi automata. We select the construction of the product Büchi net most suitable for our semidecision test. The test is based on the notion of T-invariant, and can be seen as a generalization of the ad-hoc proof method introduced and applied in [16]. We show that the test can be implemented in the framework of constraint programming [21] using the constraint programming tool 2lp [25]. Finally, we apply the test to a leader election and to a snapshot algorithm.

The paper is organised as follows. Section 2 describes the main components of the automata-theoretic approach to model-checking, tailored for the case in which the system is modelled by a Petri net. Section 3 shows how to construct the product Büchi nets. Section 4 introduces the test for net emptiness. Section 5 contains the implementation in 2lp. Section 6 is devoted to the case studies. In Section 7 we introduce a strict subclass of LTL –$LTL_{\neg}$– and a refined emptiness check. Finally, Section 8 concludes and gives an outlook. In Appendix A and B we introduce the concepts of product nets in terms of a class of high level Petri nets.

2 The automata-theoretic approach to model-checking

2.1 Transition systems

A *labelled transition system* is a fourtuple (Act, Q, Δ, q_0), where Act is an alphabet of *actions*, Q is a set of *states*, $\Delta \subseteq Q \times Act \times Q$ is a set of *transitions*, and $q_0 \in Q$ is the *initial state*.

A *full run* of a labelled transition system is an infinite sequence $q_0 a_0 q_1 a_1 q_2 \ldots$ such that $(q_i, a_i, q_{i+1}) \in \Delta$ for every $i \geq 0$. We also denote a full run by $q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} q_2 \ldots$.

When labelled transition systems are used as semantics of some process algebra only the labels of the transitions carry useful information; the intermediate states are usually irrelevant. We speak in this case of an *action-based* semantics. In action-based semantics the following definition is useful: An infinite sequence $a_0 a_1 a_2 \ldots$ of actions of $\mathcal{T}$ is an *action run* if there exists a full run $q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} q_2 \xrightarrow{a_2} \ldots$. The *action language* $L_a(\mathcal{T})$ of $\mathcal{T}$ is the set of all action runs.

When labelled transition systems are used as semantics of languages with variables, the information about the actual values of the variables is encoded into the states; the labels of the transitions are usually irrelevant. We speak in this case of a *state-based* semantics. In state-based semantics the following definition is useful: An infinite sequence $q_0 q_1 q_2 \ldots$ of states of $\mathcal{T}$ is a *state run* if there exists a full run $q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} q_2 \ldots$. The *state language* $L_s(\mathcal{T})$ of $\mathcal{T}$ is the set of all state runs.

For state-based semantics it is convenient to use (unlabelled) transition systems instead of carrying a useless action set Act around. An (unlabelled) *transition system* is a tuple (Q, Δ, q_0), where $\Delta \subseteq Q \times Q$. It can be seen as a particular case of labelled transition system in which all transitions carry the same label.

In the paper we use $L(\mathcal{T})$ to denote any of $L_a(\mathcal{T})$ or $L_s(\mathcal{T})$.

2.2 Linear-time Temporal Logic

Let Σ be a finite alphabet, and let Π be a set of propositions over Σ, i.e., a set of mappings with Σ as domain and the set $\{true, false\}$ as range. The set of formulae of *linear-time propositional temporal logic* (LTL) over the set Π is inductively defined as follows:

- if $\phi \in \Pi$ then ϕ is a formula
- if ϕ and ψ are formulae then so are $\phi \wedge \psi$, $\neg\phi$, $X\phi$ and $\phi \,\mathsf{U}\, \psi$.

We make use of the abbreviations $\phi \vee \psi = \neg(\neg\phi \wedge \neg\psi)$, $\phi \,\mathsf{V}\, \psi = \neg(\neg\phi \,\mathsf{U}\, \neg\psi)$, $\diamond\phi = \mathbf{true} \,\mathsf{U}\, \phi$ and $\Box\phi = \neg\diamond\neg\phi$. An interpretation of an LTL-formula is an infinite word $\xi \in \Sigma^\omega$. In order to formally define the satisfaction relation $\models$ of LTL, let $\xi(0)$ denote the first element of ξ, and let $\xi^{(i)}(x) = \xi(x+i)$ denote the suffix of ξ starting at position i. We have:

- $\xi \models \pi$ for $\pi \in \Pi$ if $\pi(\xi(0)) = true$.
- $\xi \models \neg\phi$ if not $\xi \models \phi$.
- $\xi \models \phi \wedge \psi$ if $\xi \models \phi$ and $\xi \models \psi$.
- $\xi \models X\phi$ if $\xi^{(1)} \models \phi$.
- $\xi \models \phi \,\mathsf{U}\, \psi$ if $\exists i \in \mathbb{N} \,:\, \xi^{(i)} \models \psi$ and $\forall j \leq i \,:\, \xi^{(j)} \models \phi$.

The *language* $L(\phi)$ of a formula ϕ over Π is the set of all words of Σ^ω that satisfy ϕ.

2.3 LTL on transition systems

We wish to use LTL to describe properties of both the action-based and the state-based semantics of a labelled transition system $\mathcal{T} = (Act, Q, \Delta, q_0)$. In the case of action-based semantics, we take $\Sigma = Act$. Π is therefore a set of propositions on the set of *actions*, and the language $L(\phi)$ of a formula ϕ is a set of action runs. We say that $\mathcal{T}$ satisfies ϕ if $L_a(\mathcal{T}) \subseteq L(\phi)$, i.e., if every action run of $\mathcal{T}$ satisfies ϕ. In state-based semantics, we take $\Sigma = Q$, and so Π is a set of propositions on the set of *states*. Analogously, we say that $\mathcal{T}$ *satisfies* ϕ if $L_s(\mathcal{T}) \subseteq L(\phi)$.

2.4 Büchi Automata

Let ϕ be a formula of LTL over a set of propositions Π. A *labelled Büchi automaton* over Π is a tuple $A = (2^\Pi, Q, \Delta, q_0, F)$, where Q is a finite set of *states*, $\Delta \subseteq Q \times 2^\Pi \times Q$ is the *transition relation*, $q_0 \in Q$ is the *initial state*, and $F \subseteq Q$ is the set of *accepting states*. An *accepting run* of A is an infinite sequence $\sigma = q_0 \Pi_0 q_1 \Pi_1 q_2 \ldots$ such that $(q_i, \Pi_i, q_{i+1}) \in \Delta$ for every $i \geq 0$, and some state of F appears infinitely often in σ. A *accepts* an infinite word $a_0 a_1 a_2 \ldots \in \Sigma^\omega$ if there exists an accepting run $q_0 \Pi_0 q_1 \Pi_1 q_2 \ldots$ such that a_i satisfies every predicate of Π_i, for every $i \geq 0$.

We define the *language* $L(A)$ of a labelled Büchi automaton A as the set of infinite words accepted by A.

We have the following important result:

Theorem 1 [32]. *Let ϕ be a formula of LTL. There exists a Büchi automaton A such that $L(\phi) = L(A)$*

In the sequel we use A_ϕ to denote a Büchi automaton satisfying $L(\phi) = L(A_\phi)$, which we assume has been constructed using some algorithm, for instance the one described in [17].

We also use *unlabelled Büchi automata*, which are tuples $A = (Q, \Delta, q_0, F)$, where $\Delta \subseteq Q \times Q$. They can be seen as a special case of labelled Büchi automata in which all transitions are labelled by the empty set of propositions.

The *nonemptiness problem* for a labelled or unlabelled Büchi automaton A consists of deciding whether $L(A)$ is nonempty. The problem is NLOGSPACE-complete [32].

2.5 Product automata

Let $\mathcal{T}_{sys}$ be a finite labelled transition system, and let ϕ be a formula over the actions or the states of $\mathcal{T}_{sys}$. The automata-based procedure to check if $\mathcal{T}_{sys}$ satisfies ϕ consists of the following steps:

- Build a labelled Büchi automaton $A_{\neg\phi}$ which accepts $L(\neg\phi)$.
- Build an unlabelled Büchi automaton A_p, called the *product* of $\mathcal{T}_{sys}$ and $A_{\neg\phi}$, which is empty iff $L(\mathcal{T}_{sys}) \cap L(\neg\phi) = \emptyset$.
- Check whether $L(A_p)$ is nonempty.

Clearly, $L(A_p)$ is empty iff $L(\mathcal{T}_{sys}) \cap L(\neg\phi) = \emptyset$ iff $L(\mathcal{T}_{sys}) \subseteq L(\phi)$ iff $\mathcal{T}_{sys}$ satisfies ϕ.

The following two subsections show how to construct A_p for action-based and state-based semantics.

Action-based semantics Let $\mathcal{T}_{sys} = (Act_{sys}, Q_{sys}, \Delta_{sys}, q_{osys})$ be a labelled transition system, and let $A_{\neg\phi} = (2^{\Pi}, Q_{\neg\phi}, \Delta_{\neg\phi}, q_{o\neg\phi}, F_{\neg\phi})$ be the labelled Büchi automaton corresponding to the negation of ϕ, where Π is a set of propositions on Act_{sys}. The product automaton of $\mathcal{T}_{sys}$ and $A_{\neg\phi}$ is the unlabelled Büchi automaton $A_p = (Q, \Delta, q_0, F)$ given by

- $Q = Q_{sys} \times Q_{\neg\phi}$,
- Δ is the smallest set such that if $(q_1, a, q_2) \in \Delta_{sys}$, $(r_1, \{\pi_1, \ldots, \pi_n\}, r_2) \in \Delta_{\neg\phi}$, and a satisfies π_i for every $1 \leq i \leq n$, then $((q_1, r_1), (q_2, r_2)) \in \Delta$.
- $q_0 = (q_{0sys}, q_{0\neg\phi})$,
- $F = Q_{sys} \times F_{\neg\phi}$.

It follows immediately from this definition that A_p is empty if and only if the set $L_a(\mathcal{T}_{sys}) \cap L(A_{\neg\phi}) = L_a(\mathcal{T}_{sys}) \cap L(\neg\phi)$ is also empty.

State-based semantics Let $\mathcal{T}_{sys} = (Q_{sys}, \Delta_{sys}, q_{0sys})$ be an unlabelled transition system, and let $A_{\neg\phi} = (2^{\Pi}, Q_{\neg\phi}, \Delta_{\neg\phi}, q_{o\neg\phi}, F_{\neg\phi})$ be the labelled Büchi automaton corresponding to the negation of ϕ, where Π is a set of propositions on Q_{sys}. The product automaton of $\mathcal{T}_{sys}$ and $A_{\neg\phi}$ is the unlabelled Büchi automaton $A_p = (Q, \Delta, q_0, F)$ given by

- $Q = Q_{sys} \times Q_{\neg\phi}$,
- Δ is the smallest set such that if $(q_1, q_2) \in \Delta_{sys}$, $(r_1, \{\pi_1, \ldots, \pi_n\}, r_2) \in \Delta_{\neg\phi}$ and q_1 satisfies π_i for every $1 \leq i \leq n$, then $((q_1, r_1), (q_2, r_2)) \in \Delta$,
- $q_0 = (q_{0sys}, q_{0\neg\phi})$,
- $F = Q_{sys} \times F_{\neg\phi}$.

The only difference with the former definition is the fact that the propositions π_i are now evaluated on the state q_1, and not on the action a.

Again, it follows immediately from this definition that A_p is empty if and only if the set $L_s(\mathcal{T}_{sys}) \cap L(A_{\neg\phi})$ is also empty.

3 Lifting the automata-theoretic model-checking method to Petri nets

3.1 Multiset Notation

We define multisets in the usual way. Let X be a set. A *multiset over* X is a mapping $\mu : Z \to \mathbb{N}$. The set theoretical operations are extended to multisets as follows:

Union:	$(\mu_1 \cup \mu_2)(a)$	$= \max(\mu_1(a), \mu_2(a))$
Intersection:	$(\mu_1 \cap \mu_2)(a)$	$= \min(\mu_1(a), \mu_2(a))$
Sum:	$(\mu_1 + \mu_2)(a)$	$= \mu_1(a) + \mu_2(a)$
Difference:	$(\mu_1 \setminus \mu_2)(a)$	$= \begin{cases} \mu_1(a) - \mu_2(a) & \text{if } \mu_1(a) \geq \mu_2(a) \\ 0 & \text{otherwise.} \end{cases}$

The set of multisets over X is denoted by $\mathcal{M}(X)$. Notions like cardinality and multiset inclusion are defined in a straightforward way.

3.2 Petri nets

We introduce a definition of Petri nets which stresses the similarities between them and labelled transition systems.

A *labelled Petri net* is a tuple $N = (Act, P, T, M_0)$ where Act is a set of *actions*, P is a finite set of *places*, $T \subseteq (\mathcal{M}(P) \times Act \times \mathcal{M}(P))$ is a set of *transitions*, and $M_0 \in \mathcal{M}(P)$ is a *marking*. For a transition $t = (P, Q)$ we sometimes call P (resp. Q) the *preset* (resp. *postset*) and write ${}^\bullet t$ (resp. $t^\bullet$). Multisets of places are called *markings*, and M_0 is called the *initial marking* of N.

A transition $t = (P_1, a, P_2)$ is *enabled at* M iff $M(p) > 0$ for every $p \in P_1$. If t is enabled at M, then t may *fire* or *occur*, yielding a new marking $M' = M - P_1 + P_2$ (where multiset addition and difference are defined in the obvious way). We denote this by $M \xrightarrow{t} M'$. A finite or infinite sequence $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} M_2 \ldots$ is called an *occurrence sequence*. $M \xrightarrow{a} M'$ for $a \in \Sigma$ denotes that there exists a transition $t = (P_1, a, P_2)$ such that $M \xrightarrow{t} M'$. A marking M is *reachable* if there exists an occurrence sequence leading to it. A labelled Petri net is *1-safe* if $M(p) \leq 1$ holds for every place p and every reachable marking M.

A *full run* of a Petri net is an infinite sequence $M_0 a_0 M_1 a_1 M_2 a_2 \ldots$ such that $M_i \xrightarrow{a_i} M_{i+1}$ for every $i \geq 0$. We also denote a full run by $M_0 \xrightarrow{a_0} M_1 \xrightarrow{a_1} M_2 \ldots$. Notice that for every full run there exists an underlying occurrence sequence.

An infinite sequence $a_0 a_1 a_2 \ldots$ of actions is an *action run* if there exists a full run $M_0 \xrightarrow{a_0} M_1 \xrightarrow{a_1} M_2 \ldots$. The *action language* $L_a(N)$ of N is the set of all action runs. An infinite sequence $M_0 M_1 M_2 \ldots$ of markings is a *state run* if there exists a full run $M_0 \xrightarrow{a_0} M_1 \xrightarrow{a_1} M_2 \ldots$. The *state language* $L_s(N)$ of N is the set of all state runs.

As usual, unlabelled Petri nets are obtained from labelled ones by dropping the labelling of transitions. So an unlabelled Petri net is a tuple (P, T, M_0) where $T \subseteq \mathcal{M}(P) \times \mathcal{M}(P)$.

If we are only interested in the structure of a Petri net, then we omit M_0 and call (P, T) just a net.

3.3 LTL on 1-safe Petri nets

We define when a 1-safe Petri net satisfies a formula of LTL. In action-based semantics Π is a set of propositions on the set of actions of the Petri net. As for transition systems, we say that a net N satisfies a formula ϕ if $L_a(N) \subseteq L(\phi)$, i.e., if every action-based run of N satisfies ϕ.

The state-based case is more interesting. For transition systems, we let Π be a set of propositions on the set of states. Since the states of a Petri net are its reachable markings, for Petri nets we should take Π as an arbitrary set of propositions on the set of *markings*. However, we restrict ourselves to propositions π_p, where p is a place of the net, with the following interpretation: a marking M satisfies π_p iff it marks the place p. We say that a net N satisfies a formula ϕ if $L_s(N) \subseteq L(\phi)$.

It is easy to see that this restriction has no important consequences: the two logics we obtain (one with arbitrary propositions over markings, the other with the restricted set), have the same expressive power for 1-safe Petri nets. To see why, suppose that a formula ϕ contains an arbitrary proposition π on markings. We can replace π by a boolean combination of propositions of the form π_p. This is at best illustrated by an example: if π is the proposition that is true of the set of markings $\{M \mid (M(p_1) = 1 \text{ and } M(p_2) = 0) \text{ or } M(p_3) = 1\}$ then it can be replaced by $(\pi_{p_1} \wedge \pi_{p_2}) \vee \pi_{p_3}$.

3.4 Büchi Nets

The product of a Büchi automaton and a 1-safe Petri net is going to be a Büchi net, the net counterpart of the unlabelled product Büchi automaton defined in Section 2.5.

A *Büchi net* is a tuple $N = (P, T, M_0, F)$, where (P, T, M_0) is an unlabelled Petri net and F is a subset of P. An *accepting run* of N is a state run $M_0 M_1 M_2 \ldots$ such that some place of F appears in infinitely many markings M_i. N is *nonempty* if it has an accepting run.

The *nonemptiness problem* for a Büchi net $N = (P, T, M_0, F)$ is the problem of deciding if N is nonempty. We have the following result:

Theorem 2. *The nonemptiness problem for 1-safe Büchi nets is PSPACE-complete.*

Proof. Let $N = (P, T, M_0, F)$ be a 1-safe Büchi net. We show that the nonemptiness problem is in NPSPACE, and then apply NPSPACE=PSPACE (Savitch's theorem). The following nondeterministic algorithm solves the nonemptiness problem using polynomial space: guess a 1-safe marking M_1 which marks F (i.e. $\sum_{p \in F'} M_1(p') > 0$). Guess (one step at a time, storing only the currently reached marking) two occurrence sequences, one that leads from M_0 to M_1 and another one (nonempty) that leads from M_1 back to M_1. Clearly, the algorithm only needs to store two markings.

To prove hardness, we provide a reduction from the problem of deciding if a linearly bounded Turing machine $\mathcal{M}$ accepts an input w. Given $\mathcal{M}$ and w, a standard construction (see for instance [10]) yields in polynomial time a 1-safe Petri net N with a distinguished place p such that $\mathcal{M}$ accepts w iff some reachable marking of N marks p. Moreover, p does not have any output transitions.

Modify the net N by adding a new transition with p as only input and output place, and transform the result into a Büchi net by taking p as (the only) final place. Clearly, $\mathcal{M}$ accepts w iff the Büchi net has an occurrence sequence that marks p infinitely often, i.e., iff N is nonempty. $\square$

3.5 Product nets in action-based semantics

It is easy to lift the definition of the product automaton to the Petri net case.

Let $N_{sys} = (Act_{sys}, P_{sys}, T_{sys}, M_{0sys})$ be a 1-safe Petri net, and let $A_{\neg\phi} = (2^{\Pi}, Q_{\neg\phi}, \Delta_{\neg\phi}, q_{o\neg\phi}, F_{\neg\phi})$ be the Büchi automaton corresponding to the negation of ϕ, where Π is a set of propositions on Act_{sys}.

Definition 3. The product Büchi net $N_p = (P, T, M_0, F)$ of N_{sys} and $A_{\neg\phi}$ is given by

- $P = P_{sys} \cup Q_{\neg\phi}$,
- T is the smallest set satisfying: if $(P_1, a, P_2) \in T_{sys}$ and $(q_1, \{\pi_1, \ldots, \pi_n\}, q_2) \in \Delta_{\neg\phi}$, and $\pi_i(a)$ holds for every $1 \leq i \leq n$, then $(P_1 + \{q_1\}, P_2 + \{q_2\})) \in T$,
- $M_0 = M_{0sys} + \{q_{0\neg\phi}\}$,
- $F = F_{\neg\phi}$.

Notice that the cardinality of the set T is at most $|T_{sys}| \cdot |\Delta_{\neg\phi}|$, and that this upper bound is reachable. The following theorem is easy to prove:

Theorem 4. *Let N_{sys} be a 1-safe Petri net, and let $A_{\neg\phi}$ be the Büchi automaton corresponding to the negation of a property ϕ. Let N_p be as in Definition 3. N_p is 1-safe and N_{sys} satisfies ϕ iff N_p is empty.*

Proof. First we show the 1-safeness of N_p. The place-vector $\mathcal{S}$ with $\mathcal{S}(p) = 1$ for all $p \in Q_{\neg\phi}$ and $\mathcal{S}(p) = 0$ else, is a covering S-invariant for the subnet generated by $A_{\neg\phi}$. Due to the 1-safeness of N_{sys} and the definition of M_0 which has weight 1 w.r.t. $\mathcal{S}$, we obtain directly the 1-safeness of N_p.
Second we show that N_{sys} does not satisfy ϕ iff N_p is nonempty.

($\Rightarrow$) Assume that N_{sys} does not satisfy ϕ, i.e., $L_a(N_{sys}) \not\subseteq L(\phi)$. Then, there exists a full run $M_0 \xrightarrow{a_0} M_1 \xrightarrow{a_1} \ldots$ of N_{sys} such that $a_0 a_1 \ldots \in L(\neg\phi)$, i.e., $a_0 a_1 \ldots$ is accepted by $A_{\neg\phi}$. Let $M_0 \xrightarrow{t_0} M_1 \xrightarrow{t_1} \ldots$ be the underlying occurrence sequence, where $t_i = (P_i, a_i, P_i')$, and let $q_0 \Pi_0 q_1 \Pi_1 \ldots$ be an accepting run of $A_{\neg\phi}$. It follows easily from the definitions that the sequence

$$M_0 + \{q_0\} \xrightarrow{(P_0+\{q_0\}, P_0'+\{q_1\})} M_1 + \{q_1\} \xrightarrow{(P_1+\{q_1\}, P_1'+\{q_2\})} \ldots$$

is a full run of N_p. Since $q_0 \Pi_0 q_1 \Pi_1 \ldots$ is an accepting run of $A_{\neg\phi}$, the sequence $(M_0 + \{q_0\})(M_1 + \{q_1\}) \ldots$ is an accepting run of N_p. So N_p is nonempty.

($\Leftarrow$) Assume that N_p is nonempty, i.e., there exists an accepting run $M_0 M_1 \ldots$ of N_p such that some place of F is infinitely often marked. Let $M_0 \xrightarrow{a_0} M_1 \xrightarrow{a_1} \ldots$ be a corresponding full run. It is easy to see that $M_0|_{P_{sys}} \xrightarrow{a_0} M_1|_{P_{sys}} \xrightarrow{a_1} \ldots$ is a full run of N_{sys}, and that $M_0|_{Q_{\neg\phi}} M_1|_{Q_{\neg\phi}} \ldots$ is an accepting run of $A_{\neg\phi}$. So $a_0 a_1 \ldots \in L_a(N_{sys}) \cap L(A_{\neg\phi})$, which implies that N_{sys} does not satisfy ϕ. □

3.6 Product nets in state-based semantics

We fix an unlabelled 1-safe Petri net $N_{sys} = (P_{sys}, T_{sys}, M_{0sys})$. We assume that the set Π of propositions on the markings of N_{sys} used to construct formulae of LTL contains only predicates π_p which hold iff the place p is marked. Clearly, we can (and will) identify the proposition π_p and the place p. With this identification, the Büchi automaton $A_{\neg\phi}$ for the negation of a formula ϕ has the form $A_{\neg\phi} = (2^{P_{sys}}, Q_{\neg\phi}, \Delta_{\neg\phi}, q_{0\neg\phi}, F_{\neg\phi})$.

Our goal is to construct a product Büchi net satisfying the following property: the product net can move from a marking (M_1, q_1) to (M_2, q_2) iff:

(1) N_{sys} can move from M_1 to M_2,
(2) there exists $(q_1, R, q_2) \in \Delta_{\neg\phi}$, and
(3) M_1 marks every place of R.

We show two different constructions. This first one is similar to that shown in Section 2.5 for transition systems. The key idea is the following: if (P_1, P_2) is a transition of the Petri net and (q_1, R, q_2) is a transition of the Büchi automaton, then we add the following transition to the product:

$$(\ P_1 + (R - P_1) + \{q_1\}\ ,\ P_2 + (R - P_1) + \{q_2\}\ )$$

It is immediate to see that this solution satisfies conditions (1) to (3) above. The product automaton can then be defined in the following way:

Definition 5. The product Büchi net $N_p = (P, T, M_0, F)$ of $N_{sys} = (P_{sys}, T_{sys}, M_{0sys})$ and $A_{\neg\phi} = (2^{P_{sys}}, Q_{\neg\phi}, \Delta_{\neg\phi}, q_{0\neg\phi}, F_{\neg\phi})$ is given by

– $P = P_{sys} \cup Q_{\neg\phi}$,

- T is the smallest set satisfying: if $(P_1, P_2) \in T_{sys}$ and $(q_1, R, q_2) \in \Delta_{\neg\phi}$, then $(P_1 + (R - P_1) + \{q_1\}, P_2 + (R - P_1) + \{q_2\}) \in T$,
- $M_0 = M_{0sys} + \{q_{0\neg\phi}\}$,
- $F = F_{\neg\phi}$.

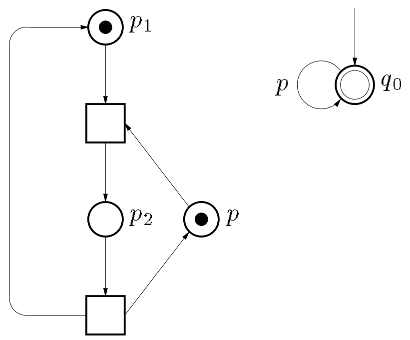


Fig.1. A Petri net N_{sys} (lhs.) and a Büchi automaton $A_{\neg\phi}$ (rhs.).

Figure 2 illustrates this definition.

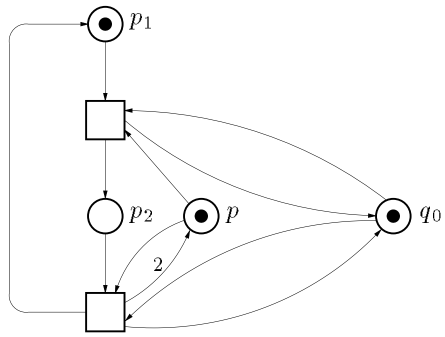


Fig.2. The product net N_p of N_{sys} and $A_{\neg\phi}$ of Figure 1 w.r.t. definition 5.

Theorem 6. *Let N_{sys} be a 1-safe Petri net, and let $A_{\neg\phi}$ be the Büchi automaton corresponding to the negation of a property ϕ. Let N_p be as in Definition 5. N_p is 1-safe and N_{sys} satisfies ϕ iff N_p is empty.*

Proof. First we show the 1-safeness of N_p. We can divide P into three sets: $P = N'_{sys} \cup (N_{sys} \setminus N'_{sys}) \cup Q_{\neg\phi}$ with $N'_{sys} = \{p \in R \mid (q, R, p) \in \Delta_{\neg\phi}\}$. The set of places $N_{sys} \setminus N'_{sys}$ is 1-safeness, due to the 1-safeness of N_{sys}. Analogous $Q_{\neg\phi}$. It remains to show that every place $p \in N'_{sys}$ is 1-safe. Due to $P_1 \cap (R - P_1) = \emptyset$ we know that every incoming arc of p has weight one. If this also holds for every outgoing arc then p is 1-safe. Otherwise the transition t which has a double weight arc to t is dead, because N_{sys} is 1-safe and every marking that enables t as transition of N_{sys} has no token on place p, but this violates that t is enabled as transition of N_p. Thus, p is 1-safe.

Second we show that N_{sys} does not satisfy ϕ iff N_p is nonempty.

($\Rightarrow$) Assume that N_{sys} does not satisfy ϕ, i.e., $L_s(N_{sys}) \not\subseteq L(\phi)$. Then there exists a state run $M_0 M_1 \ldots$ of N_{sys} which does not satisfy ϕ, and is therefore accepted by $A_{\neg\phi}$. Let $q_0 \Pi_0 q_1 \Pi_1 \ldots$ be an accepting run of $A_{\neg\phi}$. Recall that, since we have identified the propositions of LTL in the state-based case with the places of the net, a marking M satisfies a set of propositions Π iff $\Pi \subseteq M$.

Let $M_0 \xrightarrow{t_0} M_1 \xrightarrow{t_1} \ldots$ be an occurrence sequence corresponding to the state run $M_0 M_1 \ldots$, where $t_i = (P_i, Q_i)$. Due to our construction every transition t of N_{sys} "synchronizes" with every transition δ of $A_{\neg\phi}$. In particular each transition t_i synchronizes with $\delta_i = (q_i, \Pi_i, q_{i+1})$ yielding a transition $t_i' = (P_i + (\Pi_i - P_i) + \{q_i\}, Q_i + (\Pi_i - P_i) + \{q_{i+1}\})$ of N_p. Obviously, $M_i + \{q_i\}$ enables t_i' only if all places of Π_i are marked, but this is guaranteed by the condition $\Pi_i \subseteq M_i$. The occurrence of t_i' yields the marking:

$$\begin{aligned} & M_i + \{q_i\} - (P_i + (\Pi_i - P_i)) - \{q_i\} + Q_i + (\Pi_i - P_i) + \{q_{i+1}\} \\ = & M_i - P_i + Q_i + \{q_{i+1}\} \\ = & M_{i+1} + \{q_{i+1}\}. \end{aligned}$$

Accordingly, the state run $\sigma = (M_0 + \{q_0\})(M_1 + \{q_1\}) \ldots$ is an accepting run of N_p, because some final state appears infinitely often in $q_0 \Pi_0 q_1 \Pi_1 \ldots$ and thereby also in σ. So N_p has at least one accepting run, i.e., it is nonempty.

($\Leftarrow$) Analogous to the corresponding proof of Theorem 4. $\square$

Loosely speaking, in the second construction the automaton and the Petri net alternate their moves: the automaton tests if the marking M_1 marks every place of R. If this is the case, then it moves from q_1 to q_2, and transfers controls to the net, who makes its move, and transfers control back to the automaton. The alternation can be implemented by means of two scheduling places SC_1, SC_2. A token on SC_1 (SC_2) means that the automaton (the net) has to move next.

Definition 7. The product Büchi net $N_p = (P, T, M_0, F)$ of the system $N_{sys} = (P_{sys}, T_{sys}, M_{0sys})$ and the Büchi automaton $A_{\neg\phi} = (\Sigma_{\neg\phi}, Q_{\neg\phi}, \Delta_{\neg\phi}, q_{o\neg\phi}, F_{\neg\phi})$ is given by

- $P = P_{sys} \cup Q_{\neg\phi} \cup \{SC_1, SC_2\}$,
- T is the smallest set satisfying: if $(P_1, P_2) \in T_{sys}$ then $(P_1 + \{SC_2\}, P_2 + \{SC_1\}) \in T$, and if $(q_1, R, q_2) \in \Delta_{\neg\phi}$ then $(\{q_1, SC_1\} + R, \{q_2, SC_2\} + R) \in T$;
- $M_0 = M_{0sys} + \{q_{o\neg\phi}, SC_1\}$,
- $F = F_{\neg\phi}$.

See Figure 3 for an example.

Theorem 8. *Let N_{sys} be a 1-safe net system, and let $A_{\neg\phi}$ be the Büchi automaton corresponding to the negation of a property ϕ. Let N_p be as in Definition 5. N_p is 1-safe and N_{sys} satisfies ϕ iff N_p is empty.*

Proof. The 1-safeness of N_p is obvious, because N_{sys} is 1-safe and the scheduler places SC_1, SC_2 and $A_{\neg\phi}$ can be covered by an S-invariant.

($\Rightarrow$) Assume that N_{sys} does not satisfy ϕ, i.e., there exists a state run $M_0 M_1 \ldots$ of N_{sys} such that the word $M_0 M_1 \ldots$ is element of $L(\neg\phi)$. Due to Büchi's definition of $A_{\neg\phi}$ the word $M_0 M_1 \ldots$ can be accepted by $A_{\neg\phi}$. W.l.o.g. let $q_0 \Pi_0 q_1 \Pi_1 \ldots$ be the accepting run of $A_{\neg\phi}$.

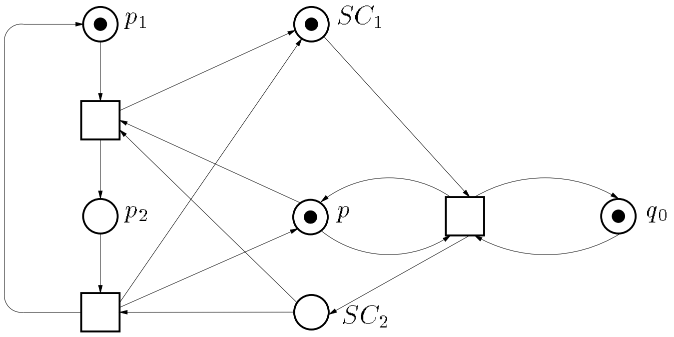


Fig.3. The product net N_p of N_{sys} and $A_{\neg\phi}$ of figure 1 w.r.t. definition 7.

Now, let $M_0 \xrightarrow{t_0} M_1 \xrightarrow{t_1} \ldots$ be the corresponding run with $t_i = (P_i, Q_i)$. Because $q_0 \Pi_0 q_1 \Pi_1 \ldots$ satisfies $M_0 M_1 \ldots$ (i.e., $\Pi_i \subseteq M_i$), a transition δ_i with preset Π_i is enabled at marking M_i. Therefore, the sequence $\sigma = M_0' \xrightarrow{\delta_0} M_0'' \xrightarrow{t_0} M_1' \xrightarrow{\delta_1} M_1'' \xrightarrow{t_1} \ldots$ with $M_i' = M_i + \{SC_1, q_i\}$ and $M_i'' = M_i + \{SC_2, q_{i+1}\}$ is a full run of N_p,because we have

(1) $M_i'|_{P_{sys}} = M_i$ and thereby $\delta_i = (\{SC_1, q_i\} + \Pi_i, \{SC_2, q_{i+1}\} + \Pi_i)$ is enabled at marking M_i',
(2) $M_i'|_{P_{sys}} = M_i''|_{P_{sys}}$ and thereby $t_i = (\{SC_2\} + P_i, \{SC_1\} + Q_i)$ is enabled at marking M_i'',
(3) $M_i' \xrightarrow{\delta_i} M_i''$, i.e.,

$$\begin{aligned} M_i'' &= M_i' - \{SC_1, q_i\} - \Pi_i + \{SC_2, q_{i+1}\} + \Pi_i \\ &= M_i + \{SC_1, q_i\} - \{SC_1, q_i\} + \{SC_2, q_{i+1}\} \\ &= M_i + \{SC_2, q_{i+1}\}, \end{aligned}$$

(4) $M_i'' \xrightarrow{t_i} M_{i+1}'$, i.e.,

$$\begin{aligned} M_{i+1}' &= M_i'' - \{SC_2\} - P_i + \{SC_1\} + Q_i \\ &= M_i + \{SC_2\} + \{q_{i+1}\} - \{SC_2\} - P_i + \{SC_1\} + Q_i \\ &= M_{i+1} + \{SC_1, q_{i+1}\}. \end{aligned}$$

Moreover σ is an accepting run of N_p, because some state of $F_{\neg\phi}$ appears infinitely often in $q_0 \Pi_0 q_1 \Pi_1 \ldots$ and thereby in σ. Thus N_p has at least one accepting run, i.e., N_p is nonempty.
($\Leftarrow$) Assume that N_p is nonempty, i.e., there exists a full run $\sigma = M_0 \xrightarrow{t_0} M_1 \xrightarrow{t_1} \ldots$ of N_p such that some place of F is marked infinitely. Because $\{SC_1, SC_2\}$ is covered by a binary P-invariant which is initially marked by only one token and every transition of N_p exchanges the token on $\{SC_1, SC_2\}$, we can divide σ up into an even subsequence σ_e of σ and an odd subsequence σ_o of σ.

On the one hand in σ_e occur only transitions of the type $(\{q_i, SC_1\} + \Pi_i, \{q_{i+1}, SC_2\} + \Pi_i)$ and therefore σ_e induces a run $q_0 \Pi_o q_1 \Pi_1 \ldots$ of $A_{\neg\phi}$. Moreover, there exists some place of F that is infinitely often marked in σ_e, because $M_{2i-1}|_F = M_{2i}|_F$. Thereby $q_0 \Pi_o q_1 \Pi_1 \ldots$ is a run accepting the word $M_0 M_2 \ldots$. Due to $\Sigma_{\neg\phi} = 2^{P_{sys}}$ the word $M_0|_{P_{sys}} M_2|_{P_{sys}} \ldots$ can also be accepted by $q_0 \Pi_o q_1 \Pi_1 \ldots$.

On the other hand in σ_o occur only transitions of the type $(P_i + \{SC_2\}, Q_i + \{SC_1\})$ and therefore σ_o induces a full run $M_1|_{P_{sys}} M_3|_{P_{sys}} \ldots$ of N_{sys}. The words $M_0|_{P_{sys}} M_2|_{P_{sys}} \ldots$ and $M_1|_{P_{sys}} M_3|_{P_{sys}} \ldots$ are equal, because

$$M_{2i+1}|_{P_{sys}} = (M_{2i} - \{SC_1, q_i\} + \{SC_2, q_{i+1}\})|_{P_{sys}} = M_{2i}|_{P_{sys}}.$$

Finally, we can conclude that there exists a full run of N_{sys} that induces a state run that can be accepted by $A_{\neg\phi}$. Thus N_{sys} violates the formula ϕ. $\square$

This second construction, contrary to the first, remains very small: its size is essentially the sum of the sizes of N_{sys} and $A_{\neg\phi}$. Unfortunately, as shown in the next section, this second construction faces other problems. We shall actually combine the two constructions in order to obtain good results.

4 Testing emptiness of Büchi nets using T-invariants

In Section 3 we have reduced the model-checking problem to the emptiness problem of Büchi nets. We now develop a semidecision test for this latter problem which avoids the construction of the reachability graph. The theory underlying the method is well-known; our contribution is a set of refinements and techniques for its application.

We have developed this test in order to verify parallel programs modelled in the language B(PN)^2 [7, 5], which are automatically translated into 1-safe Petri nets by the PEP tool [4, 1, 26]. The fact that a variable x has a value v is modelled by putting a token on a place x_v. Therefore, assertions like "the variable x takes the value 1 infinitely often" are best formalised using state-based semantics. From now on we concentrate on this semantics, but the technique is also applicable (even more easily) to the action-based case.

The test is based on the notion of T-invariant. Recall that a T-invariant of a net is a mapping $\mathcal{J}$ that assigns to each transition t a rational number $\mathcal{J}(t)$ and satisfies the following property for every place p:

$$\sum_{t \in {}^\bullet p} \mathcal{J}(t) = \sum_{t \in p^\bullet} \mathcal{J}(t)$$

T-invariants have the following fundamental property. Let M and M' be markings of a net N, and let σ be a sequence of transitions such that $M \xrightarrow{\sigma} M'$. We have $M = M'$ iff the mapping which associates to each transition t the number of times that it appears in σ is a T-invariant of N.

A T-invariant $\mathcal{J}$ of a Büchi net N is *realisable* if there exists a reachable marking M and a nonempty sequence of transitions σ such that $M \xrightarrow{\sigma} M$ and every transition t occurs exactly $\mathcal{J}(t)$ times in σ. The sequence $M \xrightarrow{\sigma} M$ is called a *realisation* of $\mathcal{J}$. Realisable T-invariants are always *semi-positive*, i.e., its components have to be nonnegative, and at least one of them must be different from 0. A T-invariant $\mathcal{J}$ is *final* if $\mathcal{J}(t) > 0$ for some transition t in the postset of a final place of N. The following result is easy to prove:

Proposition 9. *A Büchi net is nonempty iff it has a final realisable T-invariant.*

Proof. ($\Rightarrow$) If the Büchi net is nonempty, then it has an infinite occurrence sequence $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \ldots$ such that a final place p is marked at M_i for infinitely many $i \geq 0$. Since Büchi nets are 1-safe, there must be two indices $i < j$ such that $M_i = M_j$ and some marking between M_i and M_j puts a token on p. Let $\sigma = t_{i+1} \ldots t_j$, and let $\mathcal{J}$ be the mapping which assigns to a transition the number of times it occurs in σ. $\mathcal{J}$ is a final realisable T-invariant.

($\Leftarrow$) Let $M \xrightarrow{\sigma} M$ be a realisation of a final realisable T-invariant. Since M is reachable and some final place is marked along the execution of σ, the infinite sequence $M_0 \ldots (M \ldots M)^\omega$ is an accepting run. $\square$

As an immediate consequence of this proposition, if a Büchi net has no final semi-positive T-invariants, realisable or not, then it is empty. This sufficient condition for emptiness leads to a simple semidecision test, since the absence of semipositive T-invariants can be checked by solving a system of linear (in)equations of the form

$$\begin{aligned} &\mathbf{N} \cdot X = 0 \\ &X \geq 0 \\ &\textstyle\sum_{t \in F^\bullet} X(t) > 1 \end{aligned}$$

where $\mathbf{N}$ is the incidence matrix of the Büchi net, and F is the set of final places.

The practical interest of a semidecision test is directly proportional to its *quality* (i.e., how often it is successful, or, in our case, how often does it prove emptiness) and inversely proportional to its computational complexity. It is well-known that systems of linear (in)equations can be solved very efficiently using the simplex algorithm, and in guaranteed polynomial time by other techniques. So the test above is very efficient. Unfortunately, its quality is very low. In nearly all examples of interest the test fails to provide an answer even if the language of the net is empty.

A simple analysis of the T-invariants of the product Büchi nets shows that the poor quality is not surprising. The following proposition characterizes the T-invariants of the product net. For the characterization we observe that a Büchi automaton (Q, Δ, q_0, F) can also be seen as a Büchi net (P, T, M_0, F') by taking $P = Q$, $T = \Delta$, $M_0 = \{q_0\}$ and $F' = F$. So it makes sense to speak of the T-invariants of a Büchi automaton.

Proposition 10. *Let N_p be the product net of a net system N_{sys} and a Büchi automaton $A_{\neg\phi}$, obtained using Definition 5 or 7. N_p has a final semipositive T-invariant iff there exists a semipositive T-invariant J_{sys} of N_{sys} and a final semipositive T-invariant J_A of $A_{\neg\phi}$ such that*

$$\sum_{t \in T_{sys}} J_{sys}(t) = \sum_{t \in \Delta_{\neg\phi}} J_A(t)$$

Proof. First of all, let us assume arbitrary enumerations of $T_{sys} = \{t^1_{sys}, t^2_{sys}, \ldots, t^m_{sys}\}$ and $\Delta_{\neg\phi} = \{\delta^1_{\neg\phi}, \delta^2_{\neg\phi}, \ldots, \delta^n_{\neg\phi}\}$ and analogous P_{sys} and $Q_{\neg\phi}$.
(Proof w.r.t. definition 7). Let $T_p = \{t^1_{sys}, \ldots, t^m_{sys}, \delta^1_{\neg\phi}, \ldots, \delta^n_{\neg\phi}\}$ and $P_p = \{p^1_{sys}, \ldots p^{m'}_{sys}, SC_1, SC_2, q^1_{\neg\phi}, \ldots, q^{n'}_{\neg\phi}\}$. The incidence matrix of the Büchi net N_p constructed by application of definition 7 has the following scheme:

$$N_p = \begin{pmatrix} & & & 0 & \ldots & 0 \\ & N_{sys} & & \vdots & \ddots & \vdots \\ & & & 0 & \ldots & 0 \\ -1 & \ldots & -1 & 1 & \ldots & 1 \\ 1 & \ldots & 1 & -1 & \ldots & -1 \\ 0 & \ldots & 0 & & & \\ \vdots & \ddots & \vdots & & N_{\neg\phi} & \\ 0 & \ldots & 0 & & & \end{pmatrix}$$

Now let J_p be a final T-invariant of N_p:

$$N_p \cdot J_p = 0 \quad \wedge \quad \sum_{t \in F^\bullet_{\neg\phi}} J_p(t) > 1$$

$$\Leftrightarrow \quad (N_{sys}|0) \cdot J_p = 0 \quad \wedge \quad (0|N_{\neg\phi}) \cdot J_p = 0 \quad \wedge$$
$$\sum_{t \in T_{sys}} J_p(t) - \sum_{t \in \Delta_{\neg\phi}} J_p(t) = 0 \quad \wedge \quad \sum_{t \in F^{\bullet}_{\neg\phi}} J_p(t) > 1$$

If we divide J_p into two subvectors J_A and J_{sys}, i.e. $(J_{sys}|J_A)^t = J_p^t$, then we directly obtain:

$$\Leftrightarrow \quad N_{sys} \cdot J_{sys} = 0 \quad \wedge \quad N_{\neg\phi} \cdot J_A = 0 \quad \wedge$$
$$\sum_{t \in T_{sys}} J_{sys}(t) = \sum_{t \in \Delta_{\neg\phi}} J_A(t) = 0 \quad \wedge \quad \sum_{t \in F^{\bullet}_{\neg\phi}} J_A(t) > 1$$

□

(Proof w.r.t. definition 5). Let $T_p = \{t^1_{sys} \times \delta^1_{\neg\phi}, \ldots, t^m_{sys} \times \delta^1_{\neg\phi}, \ldots, t^1_{sys} \times \delta^n_{\neg\phi}, \ldots, t^m_{sys} \times \delta^n_{\neg\phi}\}$ and $P_p = \{p^1_{sys}, \ldots, p^{m'}_{sys}, q^1_{\neg\phi}, \ldots, q^{n'}_{\neg\phi}\}$. Let $N^{(i)}$ denote the i-th column of matrix N. The incidence matrix of the Büchi net N_p has the following structure:

$$N_p = \left(\begin{array}{c|c|c|c} N_{sys} & N_{sys} & \ldots & N_{sys} \\ N^{(1)}_{\neg\phi} \ldots N^{(1)}_{\neg\phi} & N^{(2)}_{\neg\phi} \ldots N^{(2)}_{\neg\phi} & \ldots & N^{(n)}_{\neg\phi} \ldots N^{(n)}_{\neg\phi} \end{array} \right)$$

Each column vector $N^{(i)}_{\neg\phi}$ appears m-times in each submatrix. Using the above-mentioned enumeration we identify the elements of J_p:

$$\begin{aligned} J_p^t = (&J_p(t^1_{sys} \times \delta^1_{\neg\phi}), \ldots, J_p(t^n_{sys} \times \delta^1_{\neg\phi}), \\ &J_p(t^1_{sys} \times \delta^2_{\neg\phi}), \ldots J_p(t^n_{sys} \times \delta^2_{\neg\phi}), \\ &\ldots, \\ &J_p(t^1_{sys} \times \delta^m_{\neg\phi}), \ldots, J_p(t^n_{sys} \times \delta^m_{\neg\phi}))^t \end{aligned}$$

Final T-invariability of J_p coincide with the following (un)-equations:

$$\begin{aligned} &(N_{sys}| \ldots |N_{sys}) \cdot J_p = 0 \quad \wedge \\ &(\underbrace{N^{(1)}_{\neg\phi} \ldots N^{(1)}_{\neg\phi}}_{m\text{-times}} | \ldots | \underbrace{N^{(n)}_{\neg\phi} \ldots N^{(n)}_{\neg\phi}}_{m\text{-times}}) \cdot J_p = 0 \quad \wedge \\ &\sum_{t \in F^{\bullet}_{\neg\phi}} J_p(t) > 1 \end{aligned}$$

Now we define J_A and J_{sys}:

$$J_A(\delta_{\neg\phi}) = \sum_{t \in T_{sys}} J(t \times \delta_{\neg\phi})$$
$$J_{sys}(t_{sys}) = \sum_{\delta \in \Delta_{\neg\phi}} J(t_{sys} \times \delta)$$

Using this definition we directly obtain:

$$\begin{aligned} &N_{sys} \cdot J_{sys} = 0 \quad \wedge \\ &N_{\neg\phi} \cdot J_A = 0 \quad \wedge \\ &\sum_{t \in F^{\bullet}_{\neg\phi}} J_A(t) > 1 \end{aligned}$$

Finally, we can conclude:

$$\begin{aligned}\sum_{t \in T_{sys}} J_{sys}(t) &= \sum_{t \in T_{sys}} \sum_{\delta \in \Delta_{\neg\phi}} J(t \times \delta) \\ &= \sum_{\delta \in \Delta_{\neg\phi}} \sum_{t \in T_{sys}} J(t \times \delta) \\ &= \sum_{\delta \in \Delta_{\neg\phi}} J_A(\delta)\end{aligned}$$

$\square$

The condition on the semi-positive T-invariants is very weak, and so it is not surprising that it is fulfilled by most Büchi nets coming from real examples, even if they do not have realisable T-invariants.

We refine Definition 5 in order to improve the quality of the test. In Section 4.1 we observe that some of the transitions of N_p can never occur. Since these transitions never appear in any infinite occurrence sequence of N_p, they can be removed without affecting Theorem 6. Clearly, after removing this transitions the resulting net has exactly the same realisable T-invariants, but *less* semipositive T-invariants, which improves the quality of the test.

Unfortunately, with the improved definition of product the number of transitions of N_p can still be unacceptably large, similarly to what happened in the action-based case. In Section 4.2 we show that this problem can be palliated by combining the improved Definition 5 with Definition 7.

4.1 Removing dead transitions

Let N_p be a product net obtained according to Definition 5, and let $t = (P_1 + (R - P_1) + \{q_1\}, P_2 + (R - P_1) + \{q_2\})$ be a transition such that there exists a place $p \in (P_2 - P_1) \cap R$. We show that t can never occur in N_p.

Let M be a marking of N_P which enables t. Then, the projection M_{sys} of M onto P_{sys} puts a token on every place of P_1 and on s. Therefore, M_{sys} enables the transition (P_1, P_2). Since $p \in P_2$, after the occurrence of the transition the place p contains two tokens. Since N_{sys} is 1-safe, M_{sys} cannot be reachable marking of N_{sys}. Since the projection on P_{sys} of a reachable marking of N_P is a reachable marking of N_{sys}, the marking M is not reachable in N_p.

This is how far we can go if we have no other information about N_{sys}. However, we often know that N_{sys} has a certain set of P-components which contain exactly one token at the initial marking. Recall that a *P-component* is a connected subnet in which every transition has exactly one input and one output place, and which is connected to other nodes of the net only through transitions[2]. The number of tokens of a P-component remains constant under the occurrence of transitions.

Information about the P-components of the net is very often available in practice. Systems modelled by 1-safe nets are usually composed by several sequential systems that communicate via message passing, rendezvous, or shared variables. In all cases, the models of these components are P-components of the global model.

Let $N_i = (P_i, T_i)$ be a P-component carrying exactly one token at the initial marking, and let

$$t = (P_1 + (R - P_1) + \{q_1\}, P_2 + (R - P_1) + \{q_2\})$$

[2] Sometimes P-components are also required to be strongly connected subnets, but that is not necessary in our case.

be a transition such that $|(P_1 + (R - P_1)) \cap P_i| > 1$. We show that t can never occur in N_p.

Let M be a marking of N_p which enables t. Since N_i is a P-component of N_{sys}, we have $|P_1 \cap P_i| \leq 1$. Therefore, either $P_1 \cap P_i = \emptyset$ and $|R \cap P_i| \geq 2$, or $p \in P_1 \cap P_i$ and $p' \in R \cap P_i$ such that $p \neq p'$. In both cases the marking M_{sys} marks at least two places of P_i, and so is not reachable in N_{sys}. Since the projection on N_{sys} of a reachable marking of N_p is a reachable marking of N_{sys}, the marking M is not reachable in N_p.

This effect can be noticed in Figure 2. The transition with the double weighted arc can be removed due to the above mentioned fact.

We introduce the following definition:

Definition 11. $(P_1, P_2) \in T_{sys}$ and $(q_1, R, q_2) \in \Delta_{\neg\phi}$ are *compatible* if the two following properties hold:

- $(P_2 \cap R) \subseteq (P_1 \cap R)$, and
- for all $1 \leq i \leq k$: if $(P_1 \cap P_i) \neq \emptyset$ and $(P_i \cap R) \neq \emptyset$, then $(P_1 \cap P_i) = (P_i \cap R)$.

If (P_1, P_2) and (q_1, R, q_2) are compatible, then we also say that (q_1, R, q_2) is compatible with (P_1, P_2), or that (P_1, P_2) is compatible with (q_1, R, q_2).

Now, in Definition 5 we can substitute the description of the set T by the following:

- T is the smallest set satisfying: if $(P_1, P_2) \in T_{sys}$ and $(q_1, R, q_2) \in \Delta_{\neg\phi}$ are compatible, then $(P_1 + (R - P_1) + \{q_1\}, P_2 + (R - P_1) + \{q_2\}) \in T$.

4.2 Combining Definition 5 and Definition 7

Let (P_1, P_2) be a transition that is compatible with every transition of $A_{\neg\phi}$. With respect to (P_1, P_2), the new definition of product coincides with the old one: the same set of transitions of the product is generated. However, n of these transitions generate $n \cdot |T_{\neg\phi}|$ transitions in the product net, which can be unacceptable if n is large.

The solution to this problem is to use the product discipline of Definition 7 for these transitions, and reserve the discipline of Definition 5 for those which can improve the quality of the test. In order to implement this idea we need the following definition:

Definition 12. A transition (P_1, P_2) of N_{sys} is *compatible* with $A_{\neg\phi}$ if it is compatible with every transition of $\Delta_{\neg\phi}$.

Definition 13. The product Büchi net $N_p = (P, T, M_0, F)$ of the system $N_{sys} = (P_{sys}, T_{sys}, M_{0sys})$ and the Büchi automaton $A_{\neg\phi} = (\Sigma_{\neg\phi}, Q_{\neg\phi}, \Delta_{\neg\phi}, q_{o\neg\phi}, F_{\neg\phi})$ is given by

- $P = P_{sys} \cup Q_{\neg\phi} \cup \{SC_1, SC_2\}$,
- T is the smallest set satisfying:
  (1) if $(q_1, R, q_2) \in \Delta_{\neg\phi}$, then $(R \cup \{q_1, SC_1\}, R \cup \{q_2, SC_2\}) \in T$,
  (2) if $(P_1, P_2) \in T_{sys}$ is compatible with $A_{\neg\phi}$, then $(P_1 + \{SC_2\}, P_2 + \{SC_1\}) \in T$,
  (3) if $(P_1, P_2) \in T_{sys}$ is not compatible with $A_{\neg\phi}$, then $(P_1 + (R - P_1) + \{q_1, SC_1\}, P_2 + (R - P_1) + \{q_2, SC_1\}) \in T$ for every $(q_1, R, q_2) \in \Delta_{\neg\phi}$ compatible with (P_1, P_2).
- $M_0 = M_{0sys} + \{q_{o\neg\phi}, SC_1\}$,
- $F = F_{\neg\phi}$.

Theorem 14. *Let N_{sys} be a 1-safe net system, and let $A_{\neg\phi}$ be the Büchi automaton corresponding to the negation of a property ϕ. Let N_p be defined as in Definition 13. N_p is 1-safe and N_{sys} satisfies ϕ iff N_p is empty.*

Proof. Due to the 1-safeness of the product net defined in Definition 5 and Definition 7, we can directly conclude that N_p is 1-safe.

It remains to show that the local behavior of transitions of T_p is equivalent to the combination of the behavior of two transitions $t_{sys} \in T_{sys}$ and $\delta_{\neg\phi} \in \Delta_{\neg\phi}$, i.e., transitions of type (1) and type (2) combine their local semantics asynchronously while transitions of type (3) are synchronizations of T_{sys} and $\Delta_{\neg\phi}$. Due to the fact that transitions of type (3) have a self loop with SC_1 these transitions cannot occur between the occurrence of a transition of type (1) and the occurrence of type (2). Thus the argumentation about full runs of N_p is analogous to the proof of Theorem 6 and Theorem 8. □

4.3 An improved Test

We have seen that a Büchi net is empty iff it has no final *realisable* T-invariants (a well known result). This result leads to a test for the emptiness problem: if a Büchi net has no semipositive final T-invariants, then it is empty. The computational complexity of the test is very good (polynomial in the size of the net), but its quality is poor.

For Büchi nets coming from the product of a Petri net and a Büchi automaton we have improved the quality by means of a refined definition of product net; unfortunately, our experiments show that the quality of the improved test is still poor, and further ideas are needed.

In this section we trade off quality for computational complexity. We introduce the notion of T*-invariant, and use it to define a new test. The new test does not have polynomial complexity anymore; it is NP-complete[3]; however, as shown later, the quality is now good enough for verifying severalinteresting liveness properties of real systems.

One of the main reasons why the test of the previous section has a low quality is the fact that the Büchi nets we wish to analyse usually contain self-loops, i.e., they contain places that are both input and output places of transitions. The presence of self-loops may lead to the typical situation shown in Figure 4. The vector $\mathcal{J} = (0,0,1,1)^t$ is a T-invariant, but not a realisable T-invariant. To prove it, observe that the subnet N' generated by the places $\{p_1, p_2\}$ and the transitions $\{t_1, \ldots t_4\}$ is a P-component (see Figure 6), and so $M(p_1) + M(p_2) = 1$ holds for every reachable marking M. Now, assume that $\mathcal{J}$ is realisable. Then it has a realisation $M \xrightarrow{\sigma} M$. Since $\mathcal{J} = (0,0,1,1)^t$, σ only contains occurrences of t_3 and t_4. It is easy to see that the projection $M' \xrightarrow{\sigma'} M'$ of $M \xrightarrow{\sigma} M$ onto the places and transitions of N' is an occurrence sequence of N'. But this leads to a contradiction: since t_3 needs a token on p_2 to occur, and t_4 needs a token on p_1, t_3 can never occur immediately after t_4; the transition t_1 must occur inbetween. Similarly, t_4 can never occur immediately after t_3; the transition t_2 must occur inbetween. More generally, the subnet of N' generated by transitions t_1 and t_2 together with their input and output places (shown in Figure 5) is *not strongly connected*, and therefore no sequence containing only t_3 and t_4 can be an occurrence sequence of N'. This shows that $\mathcal{J}$ is not realisable. In this proof we have used again information about the P-components of the net, namely the fact that N' is a P-component which carries initially one single token. This leads to the following definition:

[3] Assuming of course P≠NP.

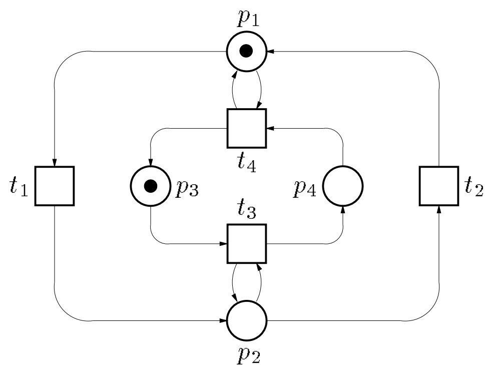


Fig. 4. Net with selfloops.

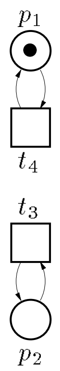


Fig. 5. The subnet N'.

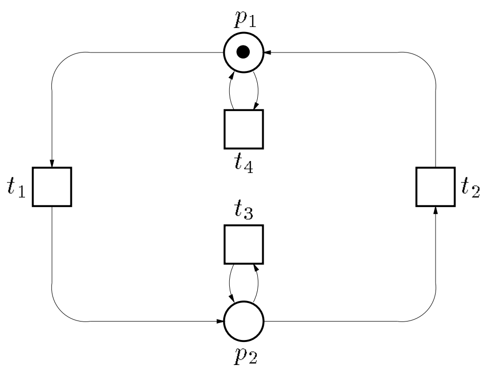


Fig. 6. The P-component with places $\{p_1, p_2\}$.

Definition 15. Let $N = (P, T)$ be a net and let $N_i = (P_i, T_i)$, $1 \leq i \leq n$ be a set of P-components of N. We call a T-vector $\mathcal{J}$ a *T*-invariant with respect to* $N_1, \ldots, N_k$ if

- $\mathcal{J}$ is a semi-positive T-invariant, and
- for every $1 \leq i \leq n$, the subnet of N_i generated by the transitions of T_i that appear in $\mathcal{J}$, together with their input and output places, is strongly connected.

The T-invariant $(0, 0, 1, 1)^t$ above is *not* a T*-invariant with respect to N', because the subnet of Figure 5 is not strongly connected.

We show that the notion of T*-invariant leads to a new emptiness test.

Lemma 16. *Realisable T-invariants are T*-invariants with respect to any set of P-components carrying one token.*

Proof. Let $\mathcal{J}$ be a realisable T-invariant of N, and let N' be an arbitrary P-component of N carrying one token at the initial marking. We show that the subnet N'' of N' generated by the transitions of N' that appear in $\mathcal{J}$, together with their input and output places, is strongly connected.

Let u and v be two arbitrary transitions of N''. Since $\mathcal{J}$ is realisable, there exists a realisation $M \xrightarrow{\sigma} M$. Since N' is a P-component, the projection $M' \xrightarrow{\sigma'} M'$ of $M \xrightarrow{\sigma} M$ on N' is an occurrence sequence of N'. Moreover, $M' \xrightarrow{\sigma'} M'$ is also an occurrence sequence of N'' because all transitions of σ' are contained in N''.

In particular, both u and v appear in σ', and so σ'^{ω} contains a subsequence of the form $u = t_0 t_1 \dots t_{n-1} t_n = v$. Since N'' is a subnet of a P-component carrying one token at the initial marking, there must be places $p_0, \dots p_{n-1}$ such that $up_0t_1p_1 \dots p_{n-2}t_{n-1}p_{n-1}v$ is a pathof N''.

Since u and v were chosen arbitrarily, any two transitions of $\mathcal{N}$ are connected by a path, which shows that N'' is strongly connected. □

Theorem 17. *Let N be a Büchi net and let N_i, $1 \leq i \leq n$ be a set of P-components of N carrying one token at the initial marking. If N has no final T^*-invariants with respect to $N_1, \dots, N_k$, then it is empty.*

Proof. By Lemma 16, if N has no final T*-invariants with respect to $N_1, \dots, N_k$, then it has no final realisable T-invariants. By Proposition 9, N is empty.

4.4 Computational complexity

We call the problem of deciding the existence of a T*-invariant for a given net and a given set of P-components the *T^*- invariant problem.*

Theorem 18. *The T^*-invariant problem is NP-complete.*

Proof. Let N be a net and let $\{N_1, \dots, N_k\}$ be a set of P-components of N. The following nondeterministic algorithm solves the T*-invariant problem in polynomial time: guess a subset T' of transitions; check for every P-component N_i that the subnet generated by the transitions of T' that belong to N_i together with their input and output places is strongly connected; check that there exists a semi-positive T-invariant $\mathcal{J}$ of N such that $\mathcal{J}(t) > 0$ iff $t \in T'$. This last part can be solved in polynomial time through reduction to a linear programming problem.

We prove NP-hardness by a reduction from the satisfiability problem for propositional formulae in conjunctive normal form (CON-SAT). An instance ϕ of CON-SAT is a conjunction of clauses $C_1, \dots, C_m$ over variables $x_1, \dots x_n$. A clause C_i is a disjunction of literals L_{ij}. A literal is either a variable x_i or its negation $\overline{x_i}$.

Given an instance of CON-SAT, we construct a Petri net $N = (P, T)$ and a set of P-components $\{N_1, \dots, N_n\}$ in polynomial time and show that N has a T*-invariant w.r.t. all N_i, iff ϕ is satisfiable.

- The set P contains the following elements:
  - for each clause C_i, $1 \leq i \leq m$, a place c_i.
  - for each variable x_i, $1 \leq i \leq n$, two places x_i and $\overline{x_i}$.
- The components P_i are defined by $P_i = \{x_i, \overline{x_i}\}$.
- The transitions in T are defined as follows:
  For each literal L_{ij} of clause C_i there exists one single transition l_{ij}. Each transition l_{ij} is connected to c_i by its preset and to $c_{i\oplus 1}$ by its postset[4]. Moreover, if L_{ij} denotes the variable x_i (negation of the variable x_i) then the transition l_{ij} is connected to place x_i ($\overline{x_i}$) by a selfloop:

$$l_{ij} = (\{c_i\} + \chi, \{c_{i\oplus 1}\} + \chi) \text{ with}$$
$$\chi = \begin{cases} x_i \text{ if } L_{ij} = x_i \\ \overline{x_i} \text{ if } L_{ij} = \overline{x_i} \end{cases}$$

[4] $\oplus$ denotes the addition modulo m.

Note that for each component N_i the places x_i and $\overline{x_i}$ are never connected. Thus, a T*-invariant does not contain two different transitions which are connected to two different places of one single component.

Now we show that ϕ is satisfiable iff the constructed Petri net N has a T*-invariant.
Assume that N has a T*-invariant $\mathcal{J}$. Due to the connection of T-invariants the tokens have to move over all places c_i. Thus, $\mathcal{J}$ has at least m positive elements, one for each clause. From the above-mentioned fact we can directly conclude that ${}^\bullet\|\mathcal{J}\| \cup \|\mathcal{J}\|^\bullet$ never contains the two places of one component.
Moreover, the assignment $\beta : \{x_1, \ldots x_n\} \to \{\textbf{true}, \textbf{false}\}$ with

$$x_i \mapsto \begin{cases} \textbf{true} & \text{if } x_i \in ({}^\bullet\|\mathcal{J}\| \cup \|\mathcal{J}\|^\bullet) \cap \bigcup\limits_{0 \le j \le n} P_j \\ \textbf{false} & \text{if } \overline{x_i} \in ({}^\bullet\|\mathcal{J}\| \cup \|\mathcal{J}\|^\bullet) \cap \bigcup\limits_{0 \le j \le n} P_j \end{cases}$$

is a valid model for ϕ, because every clause is **true** under assignment β.
Now we assume that ϕ is satisfiable. Then there must exist an assignment β that satisfies all clauses of ϕ. This means that there exists a set of literals $\mathcal{L}$ which contains for each clause at least one literal which is true under the assignment β. Now, it is obvious that the T-vector $\mathcal{J}$ with $\mathcal{J}(l_{ij}) = 1$ if $L_{ij} \in \mathcal{L}$ and $\mathcal{J}(l_{ij}) = 0$ otherwise is a T*-invariant of N. □

We illustrate this construction on an example. For this purpose we consider the formula $\phi = x_1 \wedge (x_1 \vee \overline{x_2})$. The set of P-components $\{(\{x_1, \overline{x_1}\}, \emptyset), (\{x_2, \overline{x_2}\}, \emptyset)\}$ results from the set of variables $\{x_1, x_2\}$. The corresponding composed Petri net is shown in Figure 7. The relationship between transitions (places) and

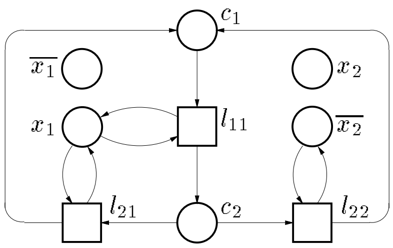


Fig.7. The corresponding net of the formula ϕ.

literals (clauses) is explained by:

$$\phi = C_1 \wedge C_2 \text{ with } C_1 = L_{11} = L_{21} = x_1, C_2 = L_{21} \vee L_{22} \text{ and } L_{22} = \overline{x_2}$$

For example, the T*-invariant $\mathcal{J}$ defined by:

$$\mathcal{J}(l_{11}) = \mathcal{J}(l_{21}) = 3$$
$$\mathcal{J}(L_{22}) = 1$$

corresponds to the assignment β:

$$\beta(x_1) = \textbf{true} \qquad \beta(x_2) = \textbf{false}$$

So the complexity of the T*-invariant problem lies betwenn the complexity of the emptiness problem, which is PSPACE-complete, and the complexity of the tests based on traditional T-invariants, which require polynomial time [14].

5 An Implementation of the T*-Invariant Test Using Constraint Programming

A system of linear inequations can be seen as a *conjunction* of linear constraints, i.e., the feasible region of the system (its set of solutions) is the set of vectors that satisfy all the constraints.

We can thus interpret linear programming as a primitive constraint programming language, in which the only available operator to combine constraints is AND. Simplex, or any other algorithm for linear programming, can be seen as an inference engine for this programming language.

While the emptiness test based on traditional T-invariants can be implemented in linear programming, this is no longer true for the T*-invariant problem: the AND construct is not powerful enough.

Fortunately, in the last years there have been a number of efforts to develop programming environments for linear and integer programming that goes well beyond the AND construct. One of these environments is 2lp [25]. Citing from [25]:"2lp is a constraint logic programming language [21, 24] with C-like syntax which can be used to make linear and integer programming part of programming in the contemporary sense of the word".

An adequate introduction to 2lp is out of the scope of this paper; we refer the interested reader to [25]. For our purposes, it suffices to know that the semantics of a 2lp program is a (not necessarily linear) *constraint* on the space of its variables, or, equivalently, a *feasible region* (the tuples of values of the variables that satisfy the constraint). 2lp contains different operators to produce complex constraints out of simpler ones. We introduce two of these operators in the following example:

$3x - 2y = 1$;
`either` $\{x + y \leq 3\}$
    `or` $\{2x - y \geq 3\}$

The operator ";" corresponds to the AND of linear programming. That is, the feasible region of the program above is the *intersection* of the feasible regions of $3x - 2y = 1$ and the `either` ... `or` constraint. The feasible region of the `either` ... `or` constraint is the *union* of the feasible regions of the constraints $x + y \leq 3$ and $2x - y \geq 3$.

2lp also provides an operator to test the consistency of sets of constraints:

$x \leq y + 3$;
$y \leq 3x - 5$;
`if not` $x = y$ `then printf(''Inconsistent'')`
`else printf(''Consistent'')`

The feasible region associated to this program is the feasible region of its first two constraints (i.e, the `not` operator does not change the feasible region). However, the `if not` ... `then` ... `else` instruction determines if the constraint $x = y$ is consistent with the first two, and answers accordingly.

We use these features to build a 2lp program that decides if a net contains a T*-invariant with respect to a set of P-components. To lighten the notation, we consider only the case in which the set contains only one component. The general case is similar.

We start by "massaging" the condition in the definition of T*-invariants concerning strong connectedness. Fix a net $N = (P, T)$ and a P-component $N' = (P', T')$ of N, and let $U \subseteq T'$. Think of U as the intersection of T' and the set of transitions of a given T-invariant, of which we would like to determine if it is also a T*-invariant. Let N'_U be the subnet of N' generated by U and $P' \cap ({}^\bullet U \cup U^\bullet)$. We wish to know whether N'_U is strongly connected or not.

Define the relation $\leadsto_U \subseteq T' \times T'$ as follows: $t \leadsto_U t'$ if $t, t' \in U$, and there exists a place $p \in P'$ such that $t \in {}^\bullet p$ and $t' \in p^\bullet$. A set $V \subseteq U$ is *closed* under $\leadsto_U$ if $t \in V$ and $t \leadsto_U t'$ implies $t' \in V$. Notice that U is trivially closed under $\leadsto_U$.

We have the following lemma:

Lemma 19. *N'_U is strongly connected iff the only nonempty subset of U that is closed under $\leadsto_U$ is U itself.*

Proof. Let $\leadsto_U^*$ denote the reflexive and transitive closure of $\leadsto_U$.

($\Rightarrow$) Let V be a nonempty subset of U closed under $\leadsto_U$, and let $t \in V$. Since N'_U is strongly connected, $t \leadsto_U^* t'$ for every transition $t' \in U$. So $U \subseteq V$.

($\Leftarrow$) If N'_U is not strongly connected, then, since N' is an S-net, there exist $t, t' \in U$ such that $t \leadsto_U t'$, but not $t' \leadsto_U^* t$. Let $V = \{t'' \mid t' \leadsto_U^* t''\}$. V is nonempty because $t' \in V$, and closed under $\leadsto_U$ (follows directly from the definition). However, $V \neq U$ because $t \in U \setminus V$. $\square$

We now define several sets of constraints on the following variables:

- A vector $\mathtt{J} \in \mathbb{Q}^{|T|}$.
- Two boolean vectors $\mathtt{U}, \mathtt{V} \in \{0, 1\}^{|T|}$

where we interpret the values of `U` and `V` as subsets of P'. Each set of constraints is to be understood conjunctively, i.e., as if its elements were linked by AND, or by the semicolon of 2lp.

(1) `J` is a semipositive T-invariant. For each $p \in P$:

$$\sum_{t \in {}^\bullet p} \mathtt{J[t]} = \sum_{t \in p^\bullet} \mathtt{J[t]}$$

and for each $t \in T$:

$$\mathtt{J[t]} \geq 0$$

(2) `J` is final.

$$\sum_{t \in F^\bullet} \mathtt{J[t]} > 0$$

(3) `U` is the intersection of T' and the support of `J`. For each $t \notin T'$:

$$\mathtt{U_i[t]} = 0$$

and for each $t \in T'$:

$$\texttt{either}\ \{\texttt{J[t]} > \texttt{0}; \texttt{U[t]} = \texttt{1}\}$$
$$\texttt{or}\ \{\texttt{J[t]} = \texttt{0}; \texttt{U[t]} = \texttt{0}\}$$

(4) `V` is a subset of `U`. For each $t \in T$:

$$\texttt{V}[t] \leq \texttt{U}[t]$$
$$\texttt{either}\ \texttt{V}[t] = 1$$
$$\texttt{or}\ \texttt{V}[t] = 0$$

(5) `V` is nonempty.

$$\sum_{t \in T} \texttt{V}[t] > 0$$

(6) `V` is closed under $\rightsquigarrow_U$. For each $t, t' \in T'$ such that there exists $p \in P'$ satisfying $t \in {}^\bullet p$ and $t' \in p^\bullet$:

$$\texttt{V}[t] + \texttt{U}[t'] \leq 1 + \texttt{V}[t']$$

(this constraint is the linear equivalent of $(t \in V \wedge t' \in U) \rightarrow t' \in V$)

(7) `V` contains less transitions than `U`.

$$\sum_{t \in T} \texttt{V}[t] < \sum_{t \in T} \texttt{U}[t]$$

Now, define the 2lp program $\mathcal{LOG}_N$ as

```
(1); (2); (3);
not {(4); (5); (6); (7)}
```

Proposition 20. *$\mathcal{LOG}_N$ is infeasible iff N contains no final T^*-invariants wrt. N'.*

Proof. The feasible region of (1) and (3) is the set of triples $(\mathcal{J}, U, V)$ where $\mathcal{J}$ is a final semipositive T-invariant and U is the intersection of T' and the support of J. The feasible region of (4) to (7) is the set of triples $(\mathcal{J}, U, V)$ where V is a proper and nonempty subset of U closed under $\rightsquigarrow_U$. According to the semantics of the `not` construct, $\mathcal{LOG}_N$ answers "No T*-invariants wrt. N'" iff the conjunction of the constraints (4) to (7) is inconsistent with the conjunction of (3) and (6). Therefore, $\mathcal{LOG}_N$ answers "No T*-invariants wrt. N'" iff for every final semipositive T-invariant the only nonempty subset of U closed under $\rightsquigarrow_U$ is U itself. This is the case iff N contains no final T*-invariants wrt. N'. □

We illustrate the construction of $\mathcal{LOG}_N$ by an example. Let us consider the Petri net given in the Figure 8. The disjunctive program $\mathcal{LOG}_N$ is given in the program notation of the CLP-tool 2lp [25]:

```
#define tr 4

continuous J[tr],
           U[tr],
```

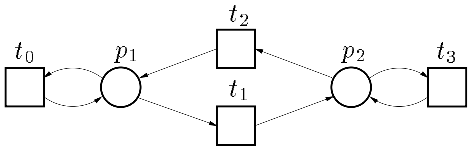


Fig.8. A P-component $N_i = (P, T)$ with $F = \{p_1\}$.

```
           V[tr];

J[2]  == J[1];
J[1]  == J[2];

and (int i=0; i < tr; i++)
    either {J[i] == 0.0; U[i] == 0.0;}
    or     {J[i] >= 1.0; U[i] == 1.0;}

J[0] + J[1] >= 1.0;

not {
     and (int i = 0; i < tr; i++)
        {V[i] <= U[i];
         either V[i] == 0.0;
         or     V[i] == 1.0;}

     V[0] + U[1] <= 1.0 + V[1];
     V[1] + U[2] <= 1.0 + V[2];
     V[1] + U[3] <= 1.0 + V[3];
     V[2] + U[0] <= 1.0 + V[0];
     V[2] + U[1] <= 1.0 + V[1];
     V[3] + U[2] <= 1.0 + V[2];

     sigma (int i = 0; i < tr; i++)
        V[i] >= 1.0;

     sigma (int i = 0; i < tr; i++) V[i]
     <=
     (sigma (int i = 0; i < tr; i++) U[i]) - 1.0;
     }
```

6 Applications

In this section we demonstrate the applicability of our verification method by means of two examples. We first consider a (variant of a) ring election algorithm designed by Chang and Roberts [12]. Then, we verify Bouge's snapshot algorithm [9]. The algorithms have been encoded in B(PN)^2 (Basic Petri Net Programming Notation) [7, 5], an imperative language designed to have a simple Petri net semantics [3, 6]. 1-safe Petri nets are then automatically generated by the PEP-tool [4, 1, 26].

6.1 A ring election algorithm

Specification Suppose a given distributed system which consists of N processes $P_0, \ldots, P_{N-1}$ that can be definitely identified by *identification numbers* $id_0, \ldots, id_{N-1}$. Moreover, the processes are connected via a token ring. The task of the ring election is the determination of an unique process, e.g. with the highest identification number.
Ring election is often needed in distributed systems without monitor in order to enable an unique master process to execute a critical operation [29].

Implementation The algorithm of Chang and Roberts [12] operates as following[5]:
Initially every process P_i sends its id_i to its right process $P_{i\oplus 1}$. Then, every process P_j that receives an id' from its left process $P_{i\ominus 1}$, transmits this id' to its right process $P_{j\oplus 1}$ if and only if this id' is greater than its own (i.e. $id' > id_j$). The process that finally receives its own identification number is definitely the master process.
In the original paper [12] a terminating algorithm is introduced. We extend it to repeated ring elections yielding a reactive system. After each ring election the master process distributes a *reset* signal to all other process restarting at the beginning. The following program presents our extension for N processes encoded in B(PN)^2.

begin

var $c_0, \ldots c_{N-1}$: **chan** 1 **of** $\{0, \ldots, N-1, reset\}$;

var $success$: $\{\mathbf{true}, \mathbf{false}\}$

proc process (
    const $id : \{0, \ldots, N-1\}$,
    ref in : **chan** 1 **of** $\{0, \ldots, N-1, reset\}$,
    ref out : **chan** 1 **of** $\{0, \ldots, N-1, reset\}$)
begin
    do
        $\langle\ out! = id\ \rangle$;
        do
            $\langle\ in? < id\ \rangle$;
            repeat
        $\square$

[5] In this context $\oplus$, resp. $\ominus$, stands for addition, resp. subtraction, modulo N.

$\langle$ *in*? $>$ *id* $\wedge$ *out*! $=$ *in*? $\wedge$ *in*? $\neq$*reset* $\rangle$;
repeat
$\Box$
$\langle$ *in*? $=$ *id* $\rangle$;
$\langle$ *success'* $=$ **true** $\rangle$;
$\langle$ *out*! $=$ *reset* $\rangle$;
$\langle$ *in*? $=$ *reset* $\rangle$;
$\langle$ *success'* $=$ **false** $\rangle$;
exit
$\Box$
$\langle$ *in*? $=$ *reset* $\wedge$ *out*! $=$ *reset* $\rangle$;
exit
od;
repeat
od
end;

$\underbrace{\text{process } (id_0, c_0, c_1)}_{P_0} \parallel \underbrace{\text{process } (id_1, c_1, c_2)}_{P_1} \parallel \ldots \parallel \underbrace{\text{process } (id_{N-1}, c_{N-1}, c_0)}_{P_{N-1}}$
end

The token ring is modeled by the fifo queues $c_0, \ldots, c_{N-1}$ such that processes P_j and $P_{j\oplus 1}$ are connected via fifo c_j. Queues have capacity one and they are only used unidirectionally. The operation $c_j!$ corresponds to a write operation in queue c_j, whereas $c_j?$ expresses a read operation from queue c_j.
We use the boolean variable *success* to indicate that at least one master process is found during a single ring election. After resetting all processes *success* is set to **false**.

Verification and results The main liveness property of the specification of the ring election is that a master process is found infinitely often. The corresponding LTL-formula is $\Box \diamond (success = \textbf{true})$. We have verified this property for $N = 1 \ldots 10$ (N is the number of processes and fifo queues). Table 6.1 summaries the sizes of the original Petri net N_{sys} and the product Büchi net N_p for some representative values of N, together with the time needed to verify the absence of T*-invariants compared to the time SPIN [20] needed to verify the property. This example is particularly favourable to our technique due to the fact that there exist no semipositive T-invariants containing transitions in the pre- or the postset of the accepting places of the underlying Büchi automaton. It must also be said that the table does not include the time needed to construct the Petri net from the B(PN)^2 program. This time was very large (about half an hour for $N = 10$), but this is due to the fact that the implementation of the PEP-compiler from B(PN)^2 into Petri nets has not been optimized yet.

6.2 A snapshot algorithm

In a distributed system each process knows only its own *local* state. However, under certain circumstances one process must be able to check the local state of all other processes – not absolutely at the same time,

[6] 128 Mbytes main memory are exceeded.

N	N_{sys}		N_p		time (sec.)	
	$\|P\|$	$\|T\|$	$\|P\|$	$\|T\|$	2lp	SPIN
5	93	91	99	96	1.24	2.20
6	117	115	123	120	2.38	9.50
7	143	141	149	146	2.44	39.40
8	171	169	177	174	3.13	(97.30)[6]
9	201	199	207	204	3.68	
10	233	231	239	236	5.01	

Table 1. Results and comparison with SPIN for Chang and Roberts' algorithm.

but each at a future point in time. This *global* state is called snapshot and makes information about *stable properties* available , e.g. termination of single processes or deadlock of the whole system.
It is possible to generate this snapshot with *or* without a monitor process [9]. We implement a simplified version with a monitor process presented in [2].

Specification Suppose a given distributed system with N processes and one single monitor process M. Every process can synchronously communicate with its neighbour processes and with the monitor process. The task of the snapshot algorithm is to enable any process at any time to initiate a snapshot that is generated in the monitor process M. After the generation of a single snapshot all processes receive it and they are reinitialized.

Implementation In [2] a method is presented to extend a given CSP-program [19] by means of certain code fragments that enable repeated snapshots. Because of lack of space we omit the explanation of the extension and refer to [9, 2] for a detailed description. However, we applied this extension to a ring architecture with 4 processes, similar to that of the ring election of the previous section. In contrast to the ring election which uses asynchronous communication via fifo queues, the snapshot algorithm of Bouge uses synchronous communication. In B(PN)^2 synchronization is modeled by channels with capacity zero. The following program describes the implementation for four processes connected via a ring:

begin

var c_0, c_1, c_2, c_3 : **chan** 0 **of** $\{data, signal\}$;
var $info, restart$: **chan** 0 **of** $\{id_0, id_1, id_2, id_3\}$;

proc process (
  const id : $\{id_0, id_1, id_2, id_3\}$,
  ref in : **chan** 0 **of** $\{data, signal\}$,
  ref out : **chan** 0 **of** $\{data, signal\}$)

begin
var *active, sent* : $\{$**true**, **false**$\}$ **init false**;

```
    do
       ⟨ active = false ⟩;
       ⟨ active' = true ⟩;
       ⟨ info! = id ⟩;
       repeat
    []
       ⟨ in? = signal ⟩;
       do
          ⟨ active = true ⟩;
          exit;
       []
          ⟨ active = false ⟩;
          ⟨ active' = true ⟩;
          ⟨ info! = id ⟩;
          exit;
       od;
       repeat
    []
       ⟨ active = true ∧ out! = signal ∧ sent = false ⟩;
       ⟨ sent' = true ⟩;
       repeat
    []
       ⟨ restart? = id ⟩;
       ⟨ active' = false ⟩;
       ⟨ sent' = false ⟩;
       repeat
    od
end;

proc monitor (
    const id_0 : {id0, id1, id2, id3},
    const id_1 : {id0, id1, id2, id3},
    const id_2 : {id0, id1, id2, id3},
    const id_3 : {id0, id1, id2, id3})

begin
var rec_0, rec_1, rec_2, rec_3, snapshot_generated : {true, false} init false;

    do
       ⟨ info? = id_0 ∧ rec_0' = true ⟩;
       repeat
    []
       ⟨ info? = id_1 ∧ rec_1' = true ⟩;
       repeat
```

```
  []
    ⟨ info? = id_2 ∧ rec_2' = true ⟩;
    repeat
  []
    ⟨ info? = id_3 ∧ rec_3' = true ⟩;
    repeat
  []
    ⟨ rec_0 = true ∧ rec_1 = true ∧ rec_2 = true ∧ rec_3 = true ⟩;
    ⟨ snapshot_generated' = true ⟩;
    ⟨ restart! = id_0 ∧ rec_0' = false ⟩;
    ⟨ restart! = id_1 ∧ rec_1' = false ⟩;
    ⟨ restart! = id_2 ∧ rec_2' = false ⟩;
    ⟨ restart! = id_3 ∧ rec_3' = false ⟩;
    ⟨ snapshot_generated' = false ⟩;
    repeat
  od
end;

process (id0, c0, c1) || process (id1, c1, c2) ||
process (id2, c2, c3) || process (id3, c3, c0) ||
monitor (id0, id1, id2, id3)
end
```

Verification and results The task of the snapshot algorithm can be specified by the following LTL-formula[7]:

$$\Box\left(\left(\bigvee_{i=0}^{3} active_i = \mathbf{true}\right) \Rightarrow \Diamond snapshot\_generated = \mathbf{true}\right)$$

The Petri net corresponding to the above-mentioned $B(PN)^2$-program has 175 places and 178 transitions. Moreover, this Petri net has 55 generating T-invariants[8]. We also tried to calculate the stubborn reduced reachability graph using INA, but there are more than 206000 reduced states (we stopped the process after 20 hours).

The synchronization with the Petri net associated with the Büchi automaton yields a product net with 179 places, 178 transitions, and 254 different[9] T-invariants that contain transitions of the pre- or the postset of the accepting places of the underlying Büchi automaton. However, if we identify the components of the Petri net corresponding the variables of the program, we see that no T-invariant satisfies the conditions of a $T^{\star}$-invariant. We can construct the product net in 80.81 seconds and check the absence of $T^{\star}$-invariants in 63.91 seconds. This example cannot[10] be verified using the SPIN-tool.

[7] Here, $active_i$ denotes the local variable of the i-th process.
[8] We tried to calculate a semipositive base using INA, but INA needs more than 8 hours.
[9] Different w.r.t. their support.
[10] 128 Mbytes main memory are exceeded.

7 Simple LTL

The semidecision test introduced in the previous sections is designed to capture the full expressiveness of LTL. One question is closely related to its design:
How can we restrict LTL in order to obtain a faster semidecision test? One variation on LTL is *simple LTL*, which contains only those formulae ϕ satisfying the following property: there exists a Büchi automaton A_ϕ which accepts $L(\phi)$ such that every circuit of A_ϕ is a self-loop [8].
We call these automata *simple* Büchi automata. Simple LTL coincides with the logic simple PLTL$_\diamond$ defined in [8] after remove its counting constraints. It is more expressive than Corbett's *ω-star-less expressions* [13], for which Corbett developed a semidecision test, because these expressions describe a subclass of simple Büchi automata.
The following remarks are devoted to introduce LTL$_\neg$ that allows a refinement of the infeasibility test which implies an impact on its performance.

First, we give a formal definition of simple LTL and the ω-regular expressions [30] that correspond to simple Büchi automata. We call these expressions *simple regular sets*. Our definition of simple regular sets is inspired by [28] and is a generalization of Corbett's star-less expressions.
In Section 2.2 we have defined LTL more abstractly in view of action based and state based semantics. Now we come off this definition and define simple LTL in a more usual way, like [28], i.e., a given set of propositions Π and thereby $\Sigma = 2^\Pi$.

Definition 21 Syntax of simple LTL. Given a finite set of propositions Π, simple LTL formulae are defined inductively as follows:

- every proposition of Π is a formula.
- if ϕ and ψ are formulae, then so are $\phi \wedge \psi$ and $\phi \vee \psi$.
- if ϕ is a formula and p is a proposition, then $p \,\mathsf{U}\, \phi$ and $\phi \,\mathsf{V}\, p$ are also formulae.
- if ϕ is a formula, then $X\phi$ is also a formula.

Definition 22 Semantics of simple LTL. The semantics of simple LTL is defined in a standard way. An interpretation of a simple LTL-formula is an infinite word ξ over the alphabet Σ, i.e. a mapping from the naturals to Σ.

- $\xi \models \pi$, if $p \in \xi(0)$ for $\pi \in \Pi$.
- $\xi \models \phi \wedge \psi$, if $\xi \models \phi$ and $\xi \models \psi$.
- $\xi \models \phi \vee \psi$, if $\xi \models \phi$ or $\xi \models \psi$.
- $\xi \models p \,\mathsf{U}\, \phi$, if $\exists i \geq 0 \,:\, \xi^{(i)} \models \phi \;\wedge\; \forall j \leq i \,:\, p \in \xi(j)$.
- $\xi \models \phi \,\mathsf{V}\, p$, if $\forall i \geq 0 \,:\, p \in \xi(i) \;\vee\; \exists j \leq i \,:\, \xi^{(j)} \models \phi$.
- $\xi \models X\phi$, if $\xi^{(1)} \models \phi$.

Note, that the usual duality of U and V is not longer valid in simple LTL in general, because $\phi \,\mathsf{V}\, p \equiv \neg(\neg\phi \,\mathsf{U}\, \neg\psi)$ is not definable in the scope of simple LTL.

For a given simple LTL formula ϕ, we denote the set of models that satisfy ϕ by $\mathcal{L}(\phi)$, i.e.

$$\mathcal{L}(\phi) = \{\xi \in \Sigma^\omega \mid \xi \models \phi\}.$$

Definition 23 Restricted regular sets [28]. Let A be a subset of Σ. We define the following restricted regular sets over Σ:

- A is a restricted regular set.
- A^*, the set of all (possibly empty) *finite* sequences over A, is a restricted regular set.
- A^ω, the set of all *infinite* sequences over A, is a restricted regular set.
- $A^\infty = A^* \cup A^\omega$ is a restricted regular set.
- Let $\mathcal{E}_1$ and $\mathcal{E}_2$ be restricted regular sets. We denote by $\mathcal{E}_1 \circ \mathcal{E}_2$ the concatenation of the two sets, defined by: $\mathcal{E}_1 \circ \mathcal{E}_2 = \{\sigma;\sigma' \mid \sigma \in \mathcal{E}_1 \cap \Sigma^* \text{ and } \sigma' \in \mathcal{E}_2\} \cup \{\sigma \mid \sigma \in \mathcal{E}_1 \cap \Sigma^\omega\}$.
  The set
  $$\coprod_{i=0}^{m} \mathcal{E}_i \stackrel{def}{=} \mathcal{E}_0 \circ \mathcal{E}_1 \circ \ldots \circ \mathcal{E}_m$$
  is a restricted regular set.

If a set A is a singleton, e.g. $A = \{a\}$ then we omit the brackets and write, for example, $a \circ b$ instead of $\{a\} \circ \{b\}$.

Definition 24 Simple regular sets. We call a restricted regular set R a Λ-set if

$$R = \coprod_{i=0}^{n} (S_i^* \cup e_i) \circ S_{n+1}^\omega$$

for some $S_i \subseteq \Sigma$ for $(1 \le i \le n+1)$ and $e_i \in \Sigma$ for $(1 \le i \le n)$, such that for all $1 \le i \le n$ either S_i is empty or e_i denotes the empty word.

A restricted regular set $\mathcal{E}$ is called a *simple regular* set if

$$\mathcal{E} = \bigcup_{j=0}^{m} R_j$$

for some Λ-sets R_i $(1 \le i \le m)$.

We call a set of sequences *simple regular definable* if there exists an equivalent simple regular set.

Theorem 25. *Given two simple regular sets $\mathcal{E}_1$ and $\mathcal{E}_2$, then $\mathcal{E}_1 \cap \mathcal{E}_2$ is simply regular definable.*

Proof. Given $\mathcal{E}_1 = \bigcup_{i=0}^{m_1} \Lambda_i$ and $\mathcal{E}_2 = \bigcup_{j=0}^{m_2} \Lambda'_j$, we construct $\mathcal{E}_1 \cap \mathcal{E}_2 = \bigcup_{i=0}^{m_1} \bigcup_{j=0}^{m_2} \Lambda_{ij}$ such that $\Lambda_i \cap \Lambda'_j = \Lambda_{ij}$. Let Λ_i and Λ'_j be Λ-sets given by

$$\Lambda_i = \coprod_{k=0}^{n_i} (S_{ik}^* \cup e_{ik}) \circ S_{n_i+1}^\omega$$
$$\Lambda'_j = \coprod_{k=0}^{n'_j} ({S'}_{jk}^* \cup {e'}_{jk}) \circ {S'}_{n'_j+1}^\omega.$$

We define Λ_{ij} inductively over n'_j:
Base Case $(n'_j = 0)$: $\Lambda_j = {S'}_{n'_j+1}^\omega$

$$\mathcal{E}_1 \cap \Lambda_j = \coprod_{k=0}^{n_i} \left(S_{ik} \cap S'_{n_j+1}\right)^* \cup \left(e_{ik} \cap S'_{n_j+1}\right) \circ (S_{n_i+1} \cap {S'}_{n'_j+1})^\omega$$

If $n'_j + 1 > 0$ we have to distinguish two cases:
If $\Lambda_j = S^* \circ \Lambda'_j$, then $\Lambda_i \cap \Lambda_j =$

$$\bigcup_{-1 \leq l \leq n_i+1} \coprod_{k=0}^{l} ((S_{ik} \cap S)^* \cup (e_{ik}) \cap S) \circ \left(\left(\left(\begin{cases} S^*_{i0} \cup e_{i0} & l = -1 \\ S^*_{il} & \begin{cases} l \neq -1 \\ \wedge \\ S_{il} \neq \emptyset \end{cases} \\ \epsilon & \text{else} \end{cases} \right\} \circ \coprod_{l+1}^{n_i} (S^*_{il} \cup e_{il}) \circ S^*_{n_i+1} \right) \cap \Lambda'_j \right) .$$

If $\Lambda_j = s \circ \Lambda'_j$, then we have to consider the first occurrence of a non-empty e_{il} in Λ_i. Therefore, let l' be the minimum of $\{l \mid l \geq 0 \wedge e_{il} \neq \epsilon\}$.
If l' is defined, then $\Lambda_i \cap \Lambda_j =$

$$\bigcup_{l=0}^{l'} (S_{il} \cap s) \circ S^*_{il} \circ \left(\left(\coprod_{k=l+1}^{n_i} (S^*_{ik} \cup e_{ik}) \circ S^\omega_{n_i+1} \right) \cap \Lambda'_j \right)$$
$$\cup\, (e_{il'} \cap s) \circ \left(\left(\coprod_{k=l'}^{n_i} (S^*_{ik} \cup e_{ik}) \circ S^\omega_{n_i+1} \right) \cap \Lambda'_j \right) .$$

If l' is undefined, then $\Lambda_i \cap \Lambda_j =$

$$\bigcup_{k=0}^{n_i+1} (S_{ik} \cap s) \circ \left(S^*_{ik} \circ \coprod_{k=l+1}^{n_i} (S^*_{ik} \cup e_{ik}) \circ S^\omega_{n_i+1} \right) \cap \Lambda'_j .$$

□

Theorem 26. *Let $p \in \Sigma$ and ϕ be a simple LTL formula. Then the following holds:*

$$\begin{aligned} \mathcal{L}(p) &= p \circ \Sigma^\omega \\ \mathcal{L}(p \,\mathsf{U}\, \phi) &= p^* \circ \mathcal{L}(\phi) \\ \mathcal{L}(X \phi) &= \Sigma \circ \mathcal{L}(\phi) \\ \mathcal{L}(\phi \,\mathsf{V}\, p) &= p^\infty \circ (p \circ \Sigma^\omega \cap \mathcal{L}(\phi)) \end{aligned}$$

Proof. We only proof the last equation. Let $\xi \in \mathcal{L}(\phi \,\mathsf{V}\, p)$, i.e. $\xi \models \phi \,\mathsf{V}\, p$.

$$\begin{aligned} &\Leftrightarrow \forall i \geq 0 \,:\, \left(p \in \xi(i) \vee \exists j \leq i \,:\, \xi^{(j)} \models \phi\right) \\ &\Leftrightarrow (\forall i \geq 0 \,:\, p \in \xi(i)) \vee \left(\exists j \leq i \,:\, \xi^{(j)} \models \phi \wedge \forall k \leq j \,:\, p \in \xi(k)\right) \\ &\Leftrightarrow \xi = p^\omega \vee \xi \in p^* \circ (p \circ \Sigma^\omega \cap \mathcal{L}(\phi)) \\ &\Leftrightarrow \xi \in p^\infty \circ (p \circ \Sigma^\omega \cap \mathcal{L}(\phi)) \end{aligned}$$

□

Corollary 27. *Let $p, q \in \Sigma$ and ϕ be a simple LTL formula. Then the following holds:*

$$\begin{aligned}
\mathcal{L}\left(p \,\mathsf{U}\, (q \,\mathsf{U}\, \phi)\right) &= p^* \circ q^* \circ \mathcal{L}(\phi) \\
\mathcal{L}\left(p \,\mathsf{U}\, (\phi \,\mathsf{V}\, q)\right) &= p^* \circ q^\infty \circ (q \circ \Sigma^\omega \cap \mathcal{L}(\phi)) \\
\mathcal{L}\left((\phi \,\mathsf{V}\, p) \,\mathsf{V}\, q\right) &= q^\infty \circ (p \cap q) \circ p^\infty \circ (p \circ \Sigma^\omega \cap \mathcal{L}(\phi)) \\
\mathcal{L}\left((p \,\mathsf{U}\, \phi) \,\mathsf{V}\, q\right) &= q^\infty \circ (q \circ \Sigma^\omega \cap p^* \circ \mathcal{L}(\phi))
\end{aligned}$$

We can summarize the last two results that show which simple LTL coincides with simple regular sets that correspond to simple Büchi automata.

Lemma 28. *For every formula ϕ is $\mathcal{L}(\phi)$ simply regular definable.*

Proof. The proof is done by induction over the structure of ϕ.
Base case: see theorem 26.
Induction Step: see corollary 27 and note that simple regular sets are closed under union and intersection (cp. theorem 25). □

Lemma 29. *For every simple regular set $\mathcal{E}$ there exists a simple LTL formula ϕ such that*

$$\mathcal{L}(\phi) = \mathcal{E}$$

Proof. Let $\mathcal{E}$ be the union of m simple regular set $\mathcal{E}_i$. Now, we define a simple LTL formula ϕ_i recursively, such that $\mathcal{E}_i = \mathcal{L}(\phi_i)$. Note that we use χ_{S_i} to denote $\bigvee_{s \in S_i} (\bigwedge_{Q \in s} Q \wedge \bigwedge_{Q \in \Pi \setminus s} \neg Q)$ (analogous χ_{e_i}).

$$\begin{aligned}
\mathcal{E}_{i,j} &= S_j^* \circ \mathcal{E}_{i,j+1} \\
\phi_{i,j} &= \chi_{S_j} \,\mathsf{U}\, \phi_{i,j+1}
\end{aligned}$$

$$\begin{aligned}
\mathcal{E}_{i,j} &= e_j \circ \mathcal{E}_{i,j+1} \\
\phi_{i,j} &= \chi_{e_j} \wedge X\,(\phi_{i,j+1})
\end{aligned}$$

$$\begin{aligned}
\mathcal{E}_{i,n+1} &= S_{n+1}^\omega \\
\phi_{i,n+1} &= \chi_{S_{n+1}} \,\mathsf{V}\, \mathbf{false}
\end{aligned}$$

Finally, we get:

$$\mathcal{L}(\phi_{1,0} \vee \phi_{2,0} \vee \ldots \vee \phi_{m,0}) = \mathcal{E}$$

□

Definition 30 Simple LTL$_\neg$. A formula ϕ belongs to *simple LTL$_\neg$* iff the negation of ϕ is a simple LTL formula, i.e. $\mathcal{L}(\neg\phi)$ is simply regular definable.

Now we can make use of the previous results yielding the modified strategy for the net-theoretic approach: For a given system N_{sys}, and a simple LTL$_\neg$ formula ϕ we construct the product net N_p and check if we can find a T-invariant of the product net that consists of exactly one transition of $A_{\neg\phi}$ corresponding to a selfloop at an accepting state. In case of simple LTL$_\neg$ a T-invariant includes only one accepting selfloop and is thereby also a T*-invariant. This fact can be used as a constraint that allows cuts in the search for such a T-invariant. Thus, in such a case we can refine the constraint (2) of $\mathcal{LOG}_N$ of Proposition 20 yielding:

(2) J is final.

$$\sum_{t\in F^{\bullet}} \texttt{J[t]} > 0 \quad \wedge \quad \sum_{t\not\in {}^{\bullet}F\cup F^{\bullet}} \texttt{J[t]} = 0$$

This refinement has an impact on the performance of the feasibility test of $\mathcal{LOG}_N$. Altough the syntax of LTL$_\neg$ is rather limited, its expressiveness still contains relevant properties:

If we consider the safety-progress classification of Chang, Manna and Pnueli [11], the classes:

- Safety,
- Guarantee,
- Obligation and Response

are included to simple LTL$_\neg$. Lamport pointed out in [23] that the relevant liveness properties have the simple structure $\Box(p \to \Diamond q)$ which as well can be expressed in simple LTL$_\neg$. Contrary, the persistence formula [11] $\Diamond\Box\neg a$ with $\mathcal{L}(\neg\Diamond\Box\neg a) = (\Sigma^* \circ a)^\omega$ cannot be expressed in simple LTL$_\neg$.

8 Conclusion

We have presented a semidecision test for the model-checking problem of 1-safe Petri nets and LTL. We make use of the automata-theoretic approach to model-checking. We have shown how to reduce the model-checking problem to the emptiness problem of a Büchi net. The test checks the presence or absence of a particular class of T-invariants which we have called T*-invariants. If no T*-invariants are present, then the Büchi net is empty, and the property holds. We were able to implement this check very easily by making use of the constraint programming tool 2lp. We have shown that there exist real algorithms for which our test allows to verify a property which cannot be proved using other exact methods.

We finish the section with some comments:

On the automata-theoretic approach to model-checking. We have shown how to lift the approach from the level of transition systems to the Petri net level. This allows to apply different methods for emptiness checking, not only semidecision tests. Wallner is working on the application of net unfoldings to the same problem [35].

On the restriction to 1-safe Petri nets. In the paper we have restricted our attention to 1-safe Petri nets. A different version of our test, however, can also be applied to arbitrary Petri nets, even unbounded ones (which is not true of the automata-theoretic approach). Essentially, instead of T-invariants it is necessary to work with so called *T-surinvariants.*

On the T*-invariant test. The test we have developed is certainly not the only possible one. We see it more as an experiment in using structural information to prove liveness properties of real examples. We have implemented some such tests in the PEP-tool, which can be applied when exact methods fail.

On the complexity of the test. It may be criticized that our test involves solving an NP-complete problem, namely the absence of T*-invariants, which may require exponential time. Actually, we think that good tests *are likely to be* NP-complete. Complexity results show that almost all interesting verification problems about 1-safe Petri nets are PSPACE-complete. Polynomial tests for such problems are bound to have very poor quality, as confirmed by our experiments. NP-complete test lie between the poor quality polynomial test and the PSPACE-complete exact methods.

On the implementation in 2lp. Linear programming plays an important rôle in net theory, but it is often too restrictive. Constraint programming tools like 2lp open a wide range of new possibilities in the application of structural objects like invariants, siphons and traps to verification problems. They also allow to implement prototypes very quickly.

Acknowledgments

We wish to thank Robert Riemann for his critical comments on an earlier version of this paper. We also benefited from numerous discussions with Frank Wallner and Ahmed Bouajjani.

A The M-net model

M-nets [6] are a class of traditional high-level Petri nets [22] with a generalized inscription of places, transitions and arcs:

$$\mathit{inscription} = (\mathit{label}, \mathit{annotation}).$$

The annotation drives the *vertical unfolding* of a M-net into a classical P/T-nets.
Moreover, the M-net model is equipped by an algebra which allows to compose smaller M-nets to more complex ones. This modular composition is called *horizontal* and is characterized by the labels of places and transitions of an M-net. The commutativity of vertical unfolding and horizontal composition is one of the major results of [6].

We now define the basic concepts of the M-net model.

A.1 Values, Variables and Bindings

Let us assume a fixed set of values VAL containing at least the distinguished element $\bullet$. Every subset of VAL describes a *type*.
Let VAR be a set of variables and OP a set of boolean and arithmetic operators, like '+', '-', '$\leq$', etc . VT denotes the set of terms (called *value terms*) built from values VAL, variables VAR and operators OP inductively in a standard way. A term without variables is called *ground term*.
A *binding* σ is a mapping $\sigma : VAR \to VAL \cup VAR$. We canonically extend bindings of variables to bindings of value terms.

A.2 Actions and Action Terms

We assume a given set ACT of *action symbols*, for short *actions*. Each action symbol $a \in ACT$ has a certain arity $ar(a)$, representing the number of parameters. Moreover, we define a *conjugation* on ACT as a bijection $^{-} : ACT \to ACT$ with the following properties: for all $a \in ACT$: $\overline{\overline{a}} = a$, $\overline{a} \neq a$, and $ar(a) = ar(\overline{a})$.
Terms $a(v_1, \ldots v_n)$ with $ar(a) = n$, $a \in ACT$ and $v_i \in VT$ are called *action terms*. AT denotes the set of all action terms. If all v_i are ground terms, then we call the action term a *ground* action.

A.3 M-nets, their Markings and the Firing Rule

Definition 31 M-net. An M-net is a triple $N = (P, T, \iota)$ with places P, transitions T and an inscription function ι which labels

- each places $p \in P$ with (λ_p, α_p) where $\lambda \in \{e, \emptyset, x\}$ is a *place label* and $\alpha_p \subseteq VAL$ is the *type* of p. The place label e (resp. x) indicates p as an entry place (resp. exit place) otherwise we call p an internal place.
- each transitions $t \in T$ with (λ_t, α_t) where λ_t is finite multiset of action terms and α_t is a finite set of value terms.
- each arc $f \in ((P \times T) \cup (T \times P))$ with a finite multiset of variables.

Moreover, there is no incoming arc to any entry place and their type is $\{\bullet\}$, similarly an exit place has no outgoing arc and its type is also $\{\bullet\}$.

An M-net is finite iff the set P of places and the set T of transitions are finite.
We define $x^\bullet = \{y \mid \iota((x, y)) \neq \emptyset\}$ and analogous ${}^\bullet x = \{y \mid \iota((y, x)) \neq \emptyset\}$, where $x, y \in P \cup T$. An empty arc inscription, i.e. $\iota((n, m)) = \emptyset$ signifies that no tokens may ever flow along that arc, i.e. there exists no effective connection along it. If we depict M-nets we will only draw arcs with non-empty arc inscriptions.
A *marking* M of an M-net (P, T, ι) is a mapping from places to finite multisets of values, respecting their type, i.e.

$$\forall v \in VAL, p \in P \; : \; v \notin \alpha_p \Rightarrow (M(p))(v) = 0.$$

Every M-net has a *standard initial marking* M_0 defined by

$$M_0(p) = \begin{cases} \{\bullet\} & \text{if } \lambda_p = e \\ \emptyset & \text{else} \end{cases}$$

with $p \in P$.

Definition 32. The transition relation is a quaternary relation $\longrightarrow \subseteq \mathcal{M} \times T \times \mathcal{B} \times \mathcal{M}$ where $\mathcal{M}$ denotes the set of all markings of N and $\mathcal{B}$ denotes the set of all bindings. A quadruple (M_1, t, σ, M_2) is defined to be in $\longrightarrow$ iff:

- σ is a binding[11] of t.
- for all $v \in \alpha_t$: $v[\sigma] = \textbf{true}$.
- There is a marking $M \in \mathcal{M}$ such that:
  - for all $p \in P : M_1(p) = \iota((p, t))[\sigma] + M(p)$
  - for all $p \in P : M_2(p) = M(p) + \iota((t, p))[\sigma]$.

A sequence $M_0 \xrightarrow{t_0} M_1 \xrightarrow{t_1} M_2 \longrightarrow \ldots \longrightarrow M_{n+1}$ is called *occurrence sequence* iff there exists bindings σ_i such that $(M_i, t_i, \sigma_i, M_{i+1})$ is in $\longrightarrow$ for each $i \leq n$.

[11] This binding has to fulfill some more properties that are not relevant in this paper. See [6] for the general definition.

A.4 Some remarks on the M-net algebra

The above-mentioned algebra on M-nets contains the classical operators known from process-algebras like CCS [27]. We need in this appendix only three of them. Let N_1, N_2 be two M-nets and $\Delta \subseteq ACT$ be a subset of action symbols:

- $N_1 \parallel N_2$ denotes the parallel composition,
- N_1 **sy** Δ synchronizes N_1 w.r.t. Δ,
- N_1 **rs** Δ restricts N_1 w.r.t. Δ,

In order to explain the semantics of **sy** and **rs** we need many auxiliary notations, that are out of the scope of our paper. The reader is referred to the original papers [3, 6]. In the next sections we show how to create an appropriate semantics of Büchi automata by means of M-nets.

B M-net Semantics of Büchi automata

Section 3 started with a Petri net N_{sys} and an LTL formula ϕ with properties over places of N_{sys}. Since markings of Petri nets put tokens on places and thereby distinguish between a marked and an unmarked place, markings of M-nets put values, e.g. natural numbers, on places and we have to differentiate between different values on one single place. Accordingly for a given M-net N_{sys} we suppose an LTL formula ϕ with properties over places *and* values.
The set of propositions of LTL has to be redefined in the context of M-nets, i.e. $\Pi = P \times VAL$ for a given M-net (P, T, ι). Intuitively a proposition $(p, v) \in \Pi$ expresses that the place p contains the value v in the current state.
Now we redefine the product net N_p for a given system N_{sys} and an LTL formula ϕ:

$$N_p = \Omega(N_{sys}, ||\phi||) \parallel \Psi(A_{\neg\phi}) \textbf{ sy } \Delta \textbf{ rs } \Delta.$$

Therefore, we have to

- define a simple transformation Ω – an addition of observer transitions that are interfaces of places appearing in propositions of ϕ.
- define an M-net semantics $\Psi(A_{\neg\phi})$ of a Büchi automaton $A_{\neg\phi}$.
- specify a special set of actions Δ and action terms for an appropriate synchronization.

Before we give a formal definition of Ω and Ψ, we want to classify our synchronization method. Godefroid has already introduced in [18] two different synchronizations of automata:

- Two automata A_1, A_2 are *synchronized on actions* iff all common actions of A_1 and A_2 are synchronized, while all other actions are interleaved.
- On the contrary A_1 and A_2 are *synchronized on states* iff the transitions of A_2 test the values of A_1 after each step of A_1.

Our synchronization is a mixed variation on both ones:

- The Büchi automaton $A_{\neg\phi}$ and all transitions of the concurrent system N_{sys} that are connected to places that are appearing in propositions of the formula ϕ are synchronized on actions.
- The Büchi automaton $A_{\neg\phi}$ and all other transitions of N_{sys} are synchronized on states.

In order to guarantee synchronization on states, i.e. each step of the system is followed by a step of the Büchi automaton, we use the same scheduler as in Section 3.

B.1 Preliminaries

Given an M-net $N = (P, T, \iota)$ we define $ac(p) \in ACT$ and $\overline{ac}(p) \in ACT$ as two monadic, conjugated action symbols for each place $p \in P$. Moreover, let $var(p) \in VAR$ be a distinct variable for each place. We distinguish two sets A_S, $A_P \subseteq ACT$ of action symbols. The set A_S consists of four unary action symbols $\{\mathbf{s}, \overline{\mathbf{s}}, \mathbf{f}, \overline{\mathbf{f}}\} = A_S$ (called *scheduler actions*) that realize the interleaving of Büchi automaton and concurrent system. The synchronization on actions is driven by the aspect A_P given by

$$A_P = \{ac(P'), \overline{ac}(P') \mid P' \subseteq P \quad \wedge \quad P' \neq \emptyset\}$$

Every action symbol $ac(P') \in A_P$ with $P' \subseteq P$ has the arity $ar(ac(P')) = |P'|$. In order to define well-formed action terms built on A_P we have to suggest an arbitrary total order $(P, \prec)$ on P.

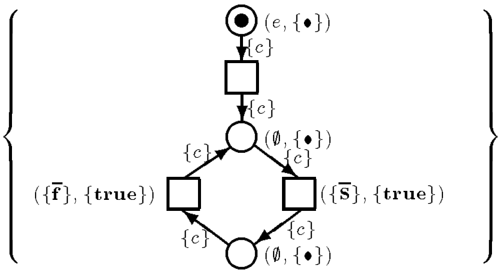
B.2 The Interfaces

Definition 33 Observer interfaces. An observer interface $Ob(p)$ for a place $p \in P$ consists of a transitions t^* and two arcs satisfying:

$$\iota((p, t^*)) = \iota((t^*, p)) = var(p) \text{ and } \iota(t^*) = (\{\overline{ac}(var(p))\}, \{\mathbf{true}\})\,.$$

Definition 34 Scheduler interface. The M-net M_S with

$M_S =$



is called the *scheduler interface* M_S.

B.3 The Transformations

Definition 35 Ω-Transformation. Let $N = (P, T, \iota)$ be an M-net. We define $\Omega'(N, P')$ with $P' \subseteq P$ as the M-net $N' \| M_S$ where M_S is the scheduler interface and N' is a slightly modification of N:

- for all $p' \in P'$ we add an observer interface $Ob(p')$.
- for all $t \in T$ where its postset and P' are disjoint we add the scheduler action $s \in A_S$ to its label, i.e. $\iota(t) = (\lambda_t + \{\mathbf{s}\}, \alpha_t)$.
- for all $t \in T$ that are connected to P' we take the variables of the outgoing arcs of t as parameters of the action term $syn(t)$. Using the total order $(P, \prec)$ we define

  $$syn(t) = ac(t^\bullet \cap P')(\mu_1, \ldots, \mu_n)$$

  where $t^\bullet \cap P' = \{p_1, \ldots, p_n\}$, $\{\mu_i\} = \iota((p_i, t))$ and $p_1 \prec \ldots \prec p_n$. Then we add this action term to the label of t:

  $$\iota(t) = (\lambda_t + syn(t), \alpha_t)$$

Definition 36 Ψ**-Transformation.** Let $A_{\neg\phi} = (\Sigma_{\neg\phi}, Q_{\neg\phi}, \Delta_{\neg\phi}, q_{0_{\neg\phi}}, F_{\neg\phi})$ be a labelled Büchi automaton. We define $\Psi(A_\phi) = (P, T, \iota)$ as the M-net semantics of $A_{\neg\phi}$ by the following rules:

- for every state $q \in Q_{\neg\phi}$ we create a place $q \in P$ with inscription $\iota(q) = (\emptyset, \{\bullet\})$,
- moreover, we add one distinguished place $p_{start} \in P$ with $\iota(p_{start}) = (e, \{\bullet\})$,
- the place p_{start} is connected to $q_{0_{\neg\phi}}$ by a distinguished transition t_{start}: $\iota((p_{start}, t_{start})) = \iota((t_{start}, q_{0_{\neg\phi}})) = c$ with $c \in VAR$,
- for every transition $t = (q_i, R_i, q_j) \in \Delta_{\neg\phi}$ we create:
  - a set of transitions

    $$\lambda(t) = \left\{ t_{P'} \mid P' \subseteq \bigcup_{(p,v) \in \mathcal{L}(t)} p \right\}$$

    with an inscription $\iota(t_{P'})$ given by

    $$\begin{aligned} \lambda_{t_{P'}} &= \overline{ac}(P')\,(var(p_1), \ldots, var(p_n)) + \{\mathbf{f}\} + \sum_{\substack{(p'',v) \in R_i, \\ p'' \notin P'}} \overline{ac}(p'')(var(p'')) \\ \alpha_{t_{P'}} &= \{var(p''') = v \mid (p''', v) \in R_i\}, \end{aligned}$$

    with $\{p_1, \ldots, p_n\} = P'$ and $p_1 \prec \ldots \prec p_n$.
  - two transitions $t'_{ij}, t''_{ij} \in T$ and a place $p_{ij} \in P$ with inscription:
    - $\iota(t'_{ij}) = (\lambda_{t'_{ij}}, \alpha_{t'_{ij}})$

      $$\begin{aligned} \lambda_{t'_{ij}} &= \{\mathbf{f}\} + \sum_{(p'',v) \in \mathcal{L}(t)} \overline{ac}(var(p'')) \\ \alpha_{t'_{ij}} &= \{var(p'') = v \mid (p'', v) \in R_i\} \end{aligned}$$

    - $\iota(t''_{ij}) = (\{\mathbf{s}\}, \{\mathbf{true}\})$,
    - $\iota(p_{ij}) = (\emptyset, \{\bullet\})$,
  - and finally we connect $\lambda(t), t'_{ij}, t''_{ij}$, and p_{ij} in the following way:
    - for each $t' \in \lambda(t)$: $\iota((p, t')) = \iota(t', p_{ij})) = \iota((p_{ij}, t''_{ij})) = \iota((t''_{ij}, q)) = c \in VAR$,
    - and $\iota((p, t'_{ij})) = \iota((t'_{ij}, q)) = c$ with $c \in VAR$.

Definition 37. Let $N_{sys} = (P, T, \iota)$ be an M-net and $A_{\neg\phi}$ be a labelled Büchi automaton. Then

$$\Omega\,(N_{sys}, \|\phi\|) \;\big\|\; \Psi\,(A_{\neg\phi}) \;\; \mathbf{sy}\; \Delta \;\mathbf{rs}\; \Delta$$

with

$$\Delta = \bigcup_{p \in \|\phi\|} \{ac(p), \overline{ac}(p)\} \cup A_S \cup \bigcup_{P' \subseteq \|\phi\|} \{ac(P'), \overline{ac}(P')\}$$

is called the *product net* N_p of N_{sys} and $A_{\neg\phi}$.

Now, we give a small example that shows the application of Ψ. Specified in LTL the formula $\phi = \diamond\Box(x = 0)$ expresses the property that eventually the value of the variable x will always be 0. Due to the duality of the modal operators, we can derive its negation, i.e. $\neg\phi = \Box \diamond \neg(x = 0) = \Box \diamond (x = 1)$. By application of classical algorithms, we get directly the Büchi automaton $A_{\neg\phi}$ depicted in Figure 9. The corresponding M-net is shown in Figure 10. Hereby, the inscription of the places and transitions is defined by:

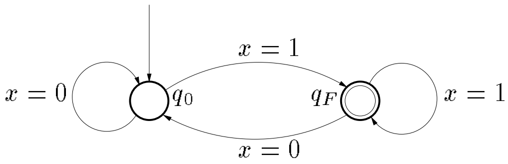


Fig.9. The corresponding Büchi automaton of the formula $\neg\phi$.

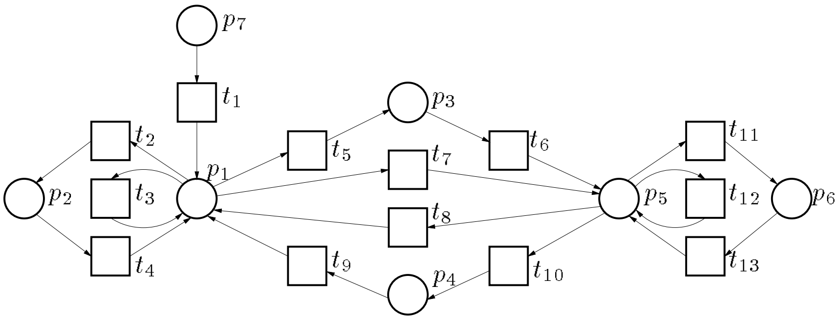


Fig.10. The corresponding M-net of the automaton $A_{\neg\phi}$.

$$
\begin{aligned}
(\emptyset, \{\bullet\} &= \iota(P_1) = \iota(P_2) = \iota(P_3) = \iota(P_4) = \iota(P_5) = \iota(P_6) \\
(e, \{\bullet\}) &= \iota(P_7) \\
(\{\mathbf{s}\}, \{\mathbf{true}\ \}) &= \iota(T_4) = \iota(T_6) = \iota(T_9) = \iota(T_{13}) \\
(\{\mathbf{f}, \overline{x}(x)\}, \{x = 0\}) &= \iota(T_3) = \iota(T_8) \\
(\{\mathbf{f}, \overline{x}(x)\}, \{x = 1\}) &= \iota(T_7) = \iota(T_{12}) \\
(\{f_x(x), \mathbf{f}\}, \{x = 1\}) &= \iota(T_5) = \iota(T_{11}) \\
(\{f_x(x), \mathbf{f}\}, \{x = 0\}) &= \iota(T_2) = \iota(T_{10}) \\
(\emptyset, \{\mathbf{true}\}) &= \iota(T_1).
\end{aligned}
$$

All arcs are inscripted by an arbitrary singleton.

References

1. Eike Best. Partial order verification with PEP. In D. Peled, G. Holzmann, and V. Pratt, editors, *Proc, POMIV'96, Partial Order Methods in Verification*. American Mathematical Society, August 1996.
2. Eike Best. *Sematics of Sequential and Parallel Programs*. Prentice Hall, 1996.
3. Eike Best, Raymond Devillers, and Jon G. Hall. The Box Calculus: A New Causal Algebra with Multi-Label Communication. In G. Rozenberg, editor, *Advances in Petri Nets 92*, volume 609 of *Lecture Notes in Computer Science*, pages 21 – 69. Springer-Verlag, 1992.
4. Eike Best and Hans Fleischhack. PEP: Programming Environment Based on Petri Nets. Hildesheimer Informatik Bericht 14/95, Universtät Hildesheim, May 1995.
5. Eike Best, Hans Fleischhack, Wojciech Fraczak, Richard Pinder Hopkins, Hanna Klaudel, and Elisabeth Pelz. An M-Net Semantics of B(PN)^2. In Jörg Desel, editor, *Structures in Concurrency Theorey (STRICT)*, pages

85 – 100. Springer, May 1995.

6. Eike Best, Hans Fleischhack, Richard Pinder Hopkins, Wojciech Fraszak, Hanna Klaudel, and Elisabeth Pelz. A Class of Composable High Level Petri Nets. In G.De Michelis and M. Diaz, editors, *Petri Nets 95*, LNCS, 1995.
7. Eike Best and Richard Pinder Hopkins. $B(PN)^2$ – a Basic Petri Net Programming Notation. In A. Bode, M. Reeve, and G. Wolf, editors, *Proceedings of PARLE '93*, volume 694 of *Lecture Notes in Computer Science*, pages 379 – 390. Springer-Verlag, 1993.
8. Ahmed Bouajjani and Peter Habermehl. Constrainted Properties, Semilinear Systems, and Petri Nets. In Ugo Montanari and Vladimiro Sassone, editors, *Proceedings of CONCUR '96*, volume 1119 of *Lecture Notes in Computer Science*, pages 481–497. Springer, 1996.
9. Luc Bouge. Repeated Synchronous Snapshots and their Implementation in CSP. In W. Brauer, editor, *Proceedings 12th ICALP*, volume 194 of *Lecture Notes in Computer Science*, pages 63 – 70. Springer, 1981.
10. Allan Chang, Javier Esparza, and Jens Palsberg. Complexity results for 1-safe nets. *Theoretical Computer Science*, 147:117 – 136, 1995.
11. Edward Chang, Zohar Manna, and Amir Pnueli. The Safety-Progress Classification. In F.L. Bauer, W. Brauer, and H. Schlichtenberg, editors, *Logic and Algebra of Specifications*, volume 94 of *NATO ASI Series: Series F Computing and System Science*, pages 143 – 202. Springer, 1993.
12. Ernest Chang and Rosemary Roberts. An Improved Algorithm for Decentralised Extrema-finding in Circular Distributed Systems. *Communication of the ACM*, 22(5):281 – 283, 1979.
13. James C. Corbett. *Automated Formal Analysis Methods for Concurrent and Real-Time Software*. PhD thesis, University of Massachusetts at Amherst, 1992.
14. Jörg Desel. Über den Beweis von Zielen mit linear-algebraischen Techniken. In J. Desel, E. Kindler, and A. Oberweis, editors, *Proceedings of the 3rd Workshop Algorithmen und Werkzeuge für Petrinetze*, Forschungsberichte, pages 8 – 13, Karlsruhe, October 1996. AIFB Universität Karlsruhe.
15. E. A. Emerson. Temporal and Modal Logic. In Jan van Leeuwen, editor, *Formel Models and Semantics*, Handbook of Theoretical Computer Science, chapter 16, pages 995 – 1072. Elsevier, 1990.
16. Javier Esparza and Glenn Bruns. Trapping Mutual Exclusion in the Box Calculus. *Theoretical Computer Science*, 153:95 – 128, 1996.
17. Rob Gerth, Doron Peled, Moshe Vardi, and Pierre Wolper. Simple On-the-fly Automatic Verification of Linear Temporal Logic. In *Protocol Specification Testing and Verification*, pages 3–18, Warsaw, Poland, 1995. Chapman & Hall.
18. Patrice Godefroid. *Partial-Order Methods for Verification of Concurrent Systems*, volume 1032 of *Lecture Notes in Computer Science*. Springer, 1996.
19. C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666 – 677, 1978.
20. Gerald J. Holzmann. *Basic Spin Manual.* AT&T Bell Laboratories, Murray Hill, New Jersey 07974.
21. Joxan Jaffar and Jean-Lois Lassez. Constraint logic programming. In *14th Annual ACM Symposium on Principles of Programming Languages*, 1987.
22. Kurt Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*, volume 1 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1992.
23. Leslie Lamport. *Sometimes* is sometimes *not ever* – On the Temporal Logic of Programs. In *Proceedings of the 7th Symposium on Principles of Programming Languages*, pages 174 – 185. ACM, 1980.
24. Brian Mayoh, Enn Tyugu, and Tarmo Uustalu. Constraint Satisfaction and Constraint Programming: A Brief Lead-In. In Brian Mayoh, Enn Tyugu, and Jaan Penjam, editors, *Advanced Study on Constraint Programming*, volume 131 of *NATO ASI Series: Series F Computer and Systems Science*, pages 1 – 16. Spinger, 1994.
25. Ken McAloon and Carol Tretkoff. *Optimization and Computational Logic*. John Wiley & Sons, 1996.

26. Stephan Melzer, Stefan Römer, and Javier Esparza. Verification using PEP. In Martin Wirsing and Maurice Nivat, editors, *Proceedings of AMAST '96*, volume 1101 of *Lecture Notes in Computer Science*, pages 591 – 594. Springer, 1996.
27. Robin Milner. *Communication and Concurrency.* Prentice-Hall, 1989.
28. A. Prasad Sistla and Lenore D. Zuck. On the Eventuality Operator in Temporal Logic. In *Logics in Computer Science.* IEEE, 1987.
29. Gerard Tel. *Introduction to distributed algorithms.* Cambridge Press, 1994.
30. Wolfgang Thomas. Automata on Infinite Objects. In Jan van Leeuwen, editor, *Formal Models and Semantics*, volume B of *Handbook of Theoretical Computer Science*, chapter 4, pages 133 – 192. Elsevier, 1990.
31. Antti Valmari. A Stubborn Attack on State Explosion. *Formal Methods in System Design*, 1:297 – 322, 1992.
32. M. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1 – 37, 1994.
33. Moshe Y. Vardi. An Automata-Theoretic Approach to Linear Temporal Logic. In Faron Moller and Graham Birtwistle, editors, *Logics for Concurrency*, volume 1043 of *Lecture Notes in Computer Science*, pages 238 – 266. Springer, 1995.
34. Moshe Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the First Symposium on Logics in Computer Science*, pages 322 – 331, Cambridge, June 1986.
35. F. Wallner. Model-Checking LTL using Net Unfoldings. Technical report, Technische Universität München, Institut für Informatik, Forthcoming 1997.