Decidability of Model Checking for Infinite-State Concurrent Systems [*]

Javier Esparza
Institut für Informatik
Technische Universität München
Arcisstr. 21 D-80290 München, Germany

Abstract

We study the decidability of the model checking problem for linear and branching time logics, and two models of concurrent computation, namely Petri nets and Basic Parallel Processes.

1 Introduction

Most techniques for the verification of concurrent systems proceed by an exhaustive traversal of the state space. Therefore, they are inherently incapable of considering systems with infinitely many states.

Recently, some new methods have been developed in order to at least palliate this problem. Using them, several verification problems for some restricted infinite-state models have been shown to be decidable. These results can be classified into those showing the decidability of equivalence relations [8, 9, 24, 26], and those showing the decidability of model checking for different modal and temporal logics. In this paper, we contribute to this second group.

The model checking problem has been studied so far for three infinite-state models: context-free processes, pushdown processes, and Petri nets. The first two are models of sequential computation, while the latter explicitely models concurrency. The modal mu-calculus, the most powerful of the modal and temporal logics commonly used for verification, is known to be decidable for context-free processes and pushdown automata. The proof is a complicated reduction to the validity problem for S2S (monadic second order logic of two successors) [32, 13]. Simpler algorithms have been given for the alternation-free fragment of the mu-calculus [5, 21]. These results have been extended to context-free like processes in [22], and to pushdown processes in [6].

The model checking problem for Petri nets was first studied by Howell and Rosier [19], who observed that a certain linear time temporal logic is undecidable even for conflict-free Petri nets, a fairly small class. This logic is interpreted on the infinite occurrence sequences of the net, and consists of atomic sentences, the usual boolean connectives, and the operator F (eventually). The atomic sentences are of type $\mathbf{ge}(\mathbf{s}, \mathbf{c})$ (with intended meaning 'at the current marking, the number of tokens on place s is

[*]This work was mostly done while the author was at the University of Edinburgh.

greater than or equal to c') or of type $\mathbf{fi(t)}$ (with intended meaning 't is the next transition in the sequence'). A Petri net is said to satisfy a formula if it has an infinite run that satisfies it.

In a subsequent paper [20], Howell, Rosier and Yen showed that the model checking problem for the positive fragment of this logic (in which negations are only applied to atomic sentences) can be reduced to the reachability problem, and is thus decidable. Jančar showed in [23] that the positive fragment with GF (infinitely often) as operator, instead of F, is decidable as well.

Although some of these results are very deep, they are also rather fragmentary. The logics have been chosen in a rather ad-hoc way, because the main interest of the authors has been to study particular problems (the main goal of [20, 23] is to study fairness problems), and not the logics themselves. In particular, branching time logics have received very little attention. Also, the logics of [19, 20, 23] contain an ad-hoc set of atomic sentences, and the impact on decidability of this or other particular choice has not been cleared up.

The goal of this paper is to offer a more systematic and global picture of the decidability issues concerning model checking infinite-state concurrent models for both linear time and branching time logics. For that, we recall some results recently obtained by the author [15], and complement them with new ones.

We consider interleaving semantics and two different models: labelled Petri nets, called just Petri nets in the rest of this section, and Basic Parallel Processes (BPPs) [7, 8]. Petri nets are a rather powerful model, which can be used to represent and analyse a large variety of systems. No natural model of concurrent computation lying strictly between Petri nets and Turing machines seems to have been proposed so far (context sensitive grammars lie strictly between Petri nets and Turing Machines [33], but they are not used as a model for concurrency). Therefore, decidability results for Petri nets are very significant, because they cannot be easily generalized, but undecidability results are not very conclusive, because a problem undecidable for arbitrary Petri nets could be decidable for relevant net classes. That is why we also study BPPs, which (in interleaving semantics) can be seen as a small subclass of Petri nets. BPPs are a rather weak process algebra, in which processes are built out of action prefix, nondeterministic choice, parallel composition without communication, and recursion. It can be argued that any reasonable infinite-state model of concurrency will have more computing power than BPPs. Therefore, undecidability results for BPPs should be expected to be very significant.

The paper is organised as follows. In the first section, we define Petri nets and BPPs. Section 2 recalls the results of [15] on linear time logics. Section 3 deals with branching time logics. In these two sections we consider action-based logics without atomic sentences. Section 4 investigates how the results change when atomic sentences are added.

2 Models: notations and basic definitions

We introduce Petri nets and Basic Parallel Processes, the two models we study. Before that, we need a few notions about transition systems.

2.1 Transition systems

A *(labelled) transition system* $\mathcal{T}$ over a set of actions *Act* consists of a set of states $\mathcal{S}$ and a relation $\xrightarrow{a} \subseteq \mathcal{S} \times \mathcal{S}$ for each action $a \in Act$. A *path* of $\mathcal{T}$ is either an infinite sequence $s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \ldots$ or a finite sequence $s_1 \xrightarrow{a_1} \ldots \xrightarrow{a_{n-1}} s_n$ such that s_n has no successors.

A transition system is *rooted* if it has a distinguished initial state. A *run* of a rooted transition system $\mathcal{T}$ is a path which starts at the initial state; a state is *reachable* if it appears in some run. The *language* of $\mathcal{T}$ is the set of sequences of actions obtained by dropping the states in the runs of $\mathcal{T}$ (so the language may contain both finite and infinite words).

In some proofs we have to consider the states of a transition system up to strong bisimulation. A relation $R \subseteq \mathcal{S} \times \mathcal{S}$ is a *strong bisimulation* if whenever $s_1 \, R \, s_2$ then, for every $a \in Act$,

- if $s_1 \xrightarrow{a} s_1'$ then $s_2 \xrightarrow{a} s_2'$ for some s_2' with $s_1' \, R \, s_2'$;
- if $s_2 \xrightarrow{a} s_2'$ then $s_1 \xrightarrow{a} s_1'$ for some s_1' with $s_2' \, R \, s_1'$.

Two states s_1 and s_2 are *strongly bisimilar*, denoted by $s_1 \sim s_2$, if there is a strong bisimulation R such that $s_1 \, R \, s_2$. This definition can be extended to states of different transition systems, by putting them 'side by side', and considering them as one single transition system.

2.2 Petri Nets

A *labelled net* N is a fourtuple (S, T, W, l), where

- S and T are two disjoint, finite sets,
- $W : (S \times T) \cup (T \times S) \to I\!N$ is a weight function, and
- l is a surjective mapping $T \to Act$, where *Act* is a set of actions (surjectivity is assumed for convenience).

The elements of S and T are called *places* and *transitions*, respectively. Places and transitions are generically called *nodes*.

A *marking* of N is a mapping $M : S \to I\!N$. A marking M *enables* a transition t if $M(s) \geq W(s,t)$ for every place s. If t is enabled at M, then it can *occur*, and its occurrence leads to the successor marking M' which is defined for every place s by

$$M'(s) = M(s) + \sum_{t \in T} (W(t,s) - W(s,t))$$

A marking M is called *dead* if it enables no transition of N.

A *(labelled) Petri net* is a pair $\Sigma = (N, M_0)$ where N is a labelled net and M_0 is a marking of N. The rooted transition system of Σ has all the markings of N as states and M_0 as initial state. For each action $a \in Act$, we define $M_1 \xrightarrow{a} M_2$ if M_1 enables some transition t labelled by a, and the marking reached by the occurrence of t is M_2. The language of a Petri net is the language of its transition system.

Unlabelled Petri nets are obtained from labelled ones by dropping the labelling function. Equivalently, one can think of unlabelled Petri nets as labelled Petri nets in which the labelling function assigns to a transition its own name. With this convention, the definition of transition system and language of a Petri net carries over to unlabelled Petri nets.

2.3 Basic and Very Basic Parallel Processes

The class of Basic Parallel Process (BPP) expressions is defined by the following abstract syntax [7, 8]:

$$
\begin{array}{rcll}
E & ::= & \mathbf{0} & \text{(inaction)} \\
 & | & X & \text{(process variable)} \\
 & | & a \cdot E & \text{(action prefix)} \\
 & | & E + E & \text{(choice)} \\
 & | & E \parallel E & \text{(merge)}
\end{array}
$$

where a belongs to a set *Act* of actions. The BPP expressions containing no occurrence of the choice operator $+$ are called Very Basic Parallel Process (VBPP) expressions.

A BPP is a family of recursive equations

$$\{X_i \stackrel{\text{def}}{=} E_i \mid 1 \leq i \leq n\}$$

where the X_i are distinct and the E_i are BPP expressions at most containing the variables $\{X_1, \ldots, X_n\}$. We further assume that every variable occurrence in the expressions E_i is *guarded*, that is, appears within the scope of an action prefix. The variable X_1 is singled out as the *leading variable* and $X_1 \stackrel{\text{def}}{=} E_1$ is called the *leading equation.*

The rooted transition system of a BPP $\{X_i \stackrel{\text{def}}{=} E_i \mid 1 \leq i \leq n\}$ has the BPP expressions over variables $X_1, \ldots, X_n$ as states; the leading variable is the initial state. For every $a \in Act$, the transition relation $\xrightarrow{a}$ is the least relation satisfying the following rules:

$$
a \cdot E \xrightarrow{a} E \qquad \frac{E \xrightarrow{a} E'}{E + F \xrightarrow{a} E'} \qquad \frac{E \xrightarrow{a} E'}{E \parallel F \xrightarrow{a} E' \parallel F}
$$

$$
\frac{E \xrightarrow{a} E'}{X \xrightarrow{a} E'} \; (X \stackrel{\text{def}}{=} E) \qquad \frac{F \xrightarrow{a} F'}{E + F \xrightarrow{a} F'} \qquad \frac{F \xrightarrow{a} F'}{E \parallel F \xrightarrow{a} E \parallel F'}
$$

For a subset of actions K, the relation $\xrightarrow{K}$ is defined as $\bigcup_{a \in K} \xrightarrow{a}$.

A BPP is in *normal form* if every expression E_i on the right hand side of an equation is of the form $a_1 \cdot \alpha_1 + \ldots + a_n \cdot \alpha_n$, where for each i the expression α_i is a merge of process variables. It is shown in [7] that every BBP is strongly bisimilar to a BPP in normal form (i.e., the leading variables are strongly bisimilar).

Every BPP in normal form can be translated into a labelled Petri net. The translation is graphically illustrated by means of an example in Figure 1. The net has a place for each variable X_i. For each subexpression $a_j \cdot \alpha_j$ in the defining equation of E_i, a transition is added having the place X_i in its preset, and the variables that appear in α_j in its postset. If a variable appears n times in α_j, then the arc leading to it is given weight n. Finally, a token is put on the place corresponding to the leading variable. It is easy to see that there exists an isomorphism between the reachable parts of the transition systems of a BPP in normal form and its associated Petri net.

Also, it follows easily from this translation that:

- the transitions of Petri nets corresponding to BPPs in normal form have exactly one input place,
- the places of VBPPs in normal form have at most one output transition, and
- all the arcs leading from places to transitions have weight 1.

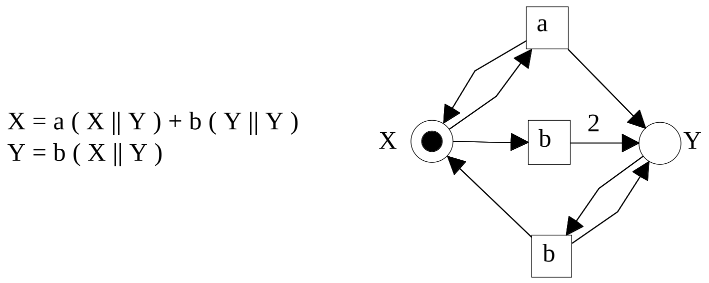

Fig. 1 A BPP and its corresponding Petri net

3 Linear Time Logics

The contents of this section are taken from [15]. We show that the model checking problem for Petri nets and closed formulas of the linear time mu-calculus is decidable. The linear time mu-calculus is a powerful linear time logic, in which all the usual linear time operators like 'always', 'eventually', and 'until' can be expressed. We describe it briefly, and refer the reader to [11] for more information.

The linear time mu-calculus has the following syntax

$$\phi ::= Z \mid \neg\phi \mid \phi \wedge \phi \mid (a)\phi \mid \nu Z.\phi$$

where a ranges over a set Act of actions, and Z over a set of propositional variables. *Free* and *bound* occurrences of variables are defined as usual. A formula is *closed* if no variable occurs free in it.

Formulas are built out of this grammar, subject to the monotonicity condition that all free occurrences of a variable Z lie in the scope of an even number of negations.

Let $Act^\star$, Act^ω be the set of finite and infinite words on Act, and let $Act^\infty = Act^\star \cup Act^\omega$. A *valuation* $\mathcal{V}$ of the logic assigns to each variable Z a set of words $\mathcal{V}(Z)$ in Act^∞. We denote by $\mathcal{V}[W/Z]$ the valuation $\mathcal{V}'$ which agrees with $\mathcal{V}$ except on Z, where $\mathcal{V}'(Z) = W$. Given a word $\sigma = a_1\, a_2 \ldots$ on Act^∞, $\sigma(1)$ denotes the first action of σ, i.e., a_1, and σ^1 denotes the word $a_2\, a_3 \ldots$. With these notations, the denotation $||\phi||_{\mathcal{V}}$ of a formula ϕ is the set of words of Act^∞ inductively defined by the following rules:

$$\begin{aligned}
||Z||_{\mathcal{V}} &= \mathcal{V}(Z)\\
||\neg\phi||_{\mathcal{V}} &= Act^\infty - ||\phi||_{\mathcal{V}}\\
||\phi \wedge \psi||_{\mathcal{V}} &= ||\phi||_{\mathcal{V}} \cap ||\psi||_{\mathcal{V}}\\
||(a)\phi||_{\mathcal{V}} &= \{\sigma \in Act^\infty \mid \sigma(1) = a \ \wedge\ \sigma^1 \in ||\phi||_{\mathcal{V}}\}\\
||\nu Z.\phi||_{\mathcal{V}} &= \cup\{W \subseteq Act^\infty \mid W \subseteq ||\phi||_{\mathcal{V}[W/Z]}\}
\end{aligned}$$

Therefore, $||\nu Z.\phi||_{\mathcal{V}}$ is the greatest fixpoint of the function which assigns to a set W of words the set $||\phi||_{\mathcal{V}[W/Z]}$.

The denotation of a closed formula ϕ is independent of the valuation; we then use the symbol $||\phi||$.

A rooted transition system $\mathcal{T}$ satisfies a closed formula ϕ of the linear time mu-calculus if every run of $\mathcal{T}$ satisfies ϕ. Accordingly, a Petri net Σ satisfies ϕ if $L(\Sigma) \subseteq ||\phi||$, where $L(\Sigma)$ denotes the language of its transition system. Notice that, with this definition of satisfaction, it can be the case that Σ satisfies neither a formula nor its negation.

According to our definition, the denotation of a formula is a set of finite and infinite words. This choice is due to the fact that we wish to define satisfaction as $L(\Sigma) \subseteq ||\phi||$, and we wish to define $L(\Sigma)$ as a set of finite and infinite words, in order to account for deadlock properties. In particular, with this definition we can express deadlock-freedom as

$$\nu Z. \bigvee_{a \in Act} (a)Z$$

Sometimes the denotation of a formula is defined as a set of only infinite words, as for instance in [11].

We define the model checking problem for the linear time mu-calculus and Petri nets as follows: given a Petri net Σ and a closed formula ϕ, determine if Σ satisfies ϕ or not. This problem is proved to be decidable in [15]. Here we just sketch the proof. The decision procedure is based on an automata-theoretic characterisation of the logic [35]. It is shown in [11] that, given a closed formula ϕ, there exists a finite automaton $A_{\neg\phi}$ and a Büchi automaton $B_{\neg\phi}$ that accept the finite and infinite words of $||\neg\phi||$, respectively. It is then easy to see that Σ satisfies ϕ if and only if $L(\Sigma) \cap L(A_{\neg\phi}) = \emptyset$ and $L(\Sigma) \cap L(B_{\neg\phi}) = \emptyset$.

In order to decide these two properties, two new Petri nets $\Sigma \times \Sigma^{A_{\neg\phi}}$ and $\Sigma \times \Sigma^{B_{\neg\phi}}$ are constructed, having the properties $L(\Sigma \times \Sigma^{A_{\neg\phi}}) = L(\Sigma) \cap L(\Sigma^{A_{\neg\phi}})$, and $L(\Sigma \times \Sigma^{B_{\neg\phi}}) = L(\Sigma) \cap L(\Sigma^{B_{\neg\phi}})$. Then, the two following results are proved:

- $L(\Sigma) \cap L(B_\phi) \neq \emptyset$ holds if and only if the Petri net $\Sigma \times \Sigma^{B_\phi}$ has a run which contains infinitely many occurrences of transitions of a set T. The set T can be efficiently computed from Σ and B_ϕ.
- $L(\Sigma) \cap L(A_\phi) \neq \emptyset$ if and only if the Petri net $\Sigma \times \Sigma^{A_\phi}$ has a reachable dead marking which marks some place of a set S. The set S can be efficiently computed from Σ and B_ϕ.

The existence of the run was shown to be decidable by Jantzen and Valk [25]. Yen shows in [36] that it is decidable within exponential space.

The existence of the dead marking can be decided by solving an exponential number of instances of the reachability problem. The reachability problem is decidable [27, 30] and known to require exponential space [29], but none of the algorithms known so far is primitive recursive.

4 Branching Time Logics

The decidability border changes rather drastically when we move from linear time to branching time logics. We have seen that a powerful linear time logic like the linear time mu-calculus is decidable for a powerful model like Petri nets. In this section we shall see that a weak branching time logic ise undecidable even for VBPPs, a very weak model of computation.

For a smooth approach to the result, we shall first prove that the modal mu-calculus is undecidable for VBPPs. We introduce the modal mu-calculus very briefly, more details can be found in [34].

4.1 The modal mu-calculus

The modal mu-calculus has the same syntax as the linear time mu-calculus, just substituting $\langle a\rangle\phi$ for $(a)\phi$. It is interpreted on labelled transition systems. Let $\mathcal{T} = (\mathcal{S}, \xrightarrow{a}_{a\in Act})$ be a labelled transition system. Given a valuation $\mathcal{V}$ that assigns to each variable a subset of states $\mathcal{S}$, the denotation of a formula is a subset of states defined by the following rules:

$$\begin{aligned}
\|Z\|_{\mathcal{V}} &= \mathcal{V}(Z)\\
\|\neg\phi\|_{\mathcal{V}} &= \mathcal{S} - \|\phi\|_{\mathcal{V}}\\
\|\phi_1 \wedge \phi_2\|_{\mathcal{V}} &= \|\phi_1\|_{\mathcal{V}} \cap \|\phi_2\|_{\mathcal{V}}\\
\|\langle a\rangle\phi\|_{\mathcal{V}} &= \{s \in \mathcal{S} \mid \exists s' \in \mathcal{S}. s \xrightarrow{a} s' \wedge s' \in \|\phi\|_{\mathcal{V}}\}\\
\|\nu Z.\phi\|_{\mathcal{V}} &= \bigcup\{A \subseteq \mathcal{S} \mid A \subseteq \|\phi\|_{\mathcal{V}[A/Z]}\}
\end{aligned}$$

Loosely speaking, a state belongs to $\|\langle a\rangle\phi\|_{\mathcal{V}}$ if it has some succesor, reachable by means of an a action, which satisfies ϕ. We use the following abbreviations:

$$\begin{array}{cc}
\mathbf{true} = \nu Z.Z & \phi_1 \vee \phi_2 = \neg(\neg\phi_1 \wedge \neg\phi_2)\\
\mu Z.\phi = \neg\nu Z.\neg\phi[\neg Z/Z] & [a]\phi = \neg\langle a\rangle\neg\phi
\end{array}$$

As in the case of the linear time mu-calculus, the denotation of a closed formula is independent of the valuation, and then we drop the index ν. Observe that $\|\mathbf{true}\| = \mathcal{S}$.

A rooted transition system satisfies a closed formula ϕ if the denotation of ϕ contains the initial state.

The modal mu-calculus has the following property: if two states of a transition system are strongly bisimilar, then they satisfy exactly the same closed formulas of the logic.

4.2 Undecidability of the modal mu-calculus for VBPPs

We prove the undecidability of the model checking problem by means of a reduction from the halting problem of counter machines [31] (more precisely, to the halting problem for counter machines whose counters are initialised to 0).

A *counter machine* $\mathcal{M}$ is a tuple

$$(\{q_0, \ldots, q_{n+1}\}, \{c_1, \ldots, c_m\}, \{\delta_0, \ldots, \delta_n\})$$

where c_i are the *counters*, q_i are the *states* with q_0 being the *initial state* and q_{n+1} the unique *halting state*, and δ_i is the *transition rule* for state q_i $(0 \le i \le n)$. The states are divided into two types. The states of type I have transition rules of the form

$$c_j := c_j + 1;\ \text{goto}\ q_k$$

for some j, k. The states of type II have transition rules of the form

$$\text{if}\ c_j = 0\ \text{then goto}\ q_k\ \text{else}\ (c_j := c_j - 1;\ \text{goto}\ q_{k'})$$

for some j, k, k'.

A *configuration* of $\mathcal{M}$ is a tuple $(q_i, j_1, \ldots, j_m)$, where q_i is a state, and $j_1, \ldots, j_m$ are natural numbers indicating the contents of the counters. We are interested in the behaviour of a machine whose counters are initialised to 0. Therefore, we define the *initial configuration* as $(q_0, 0, \ldots, 0)$. The *computation* of $\mathcal{M}$ is the sequence of configurations which starts with the initial configuration and is inductively defined in the expected way, according to the transition rules. Notice that the computation of $\mathcal{M}$ is unique, because each state has at most one transition rule. We say that $\mathcal{M}$ *halts* if its computation is finite, or, equivalently, if its computation contains a configuration with state q_{n+1}. It is undecidable if a given counter machine halts [31].

Given a counter machine $\mathcal{M}$, our reduction constructs a VBPP with leading variable M, and a formula *Halt* of the modal mu-calculus such that $\mathcal{M}$ halts if and only if M satisfies *Halt*.

If instead of VBPPs we were considering a Turing-powerful model like CCS, the problem would be trivial: M would just be a faithful model of the counter machine $\mathcal{M}$, in which the occurrence of an action halt signals termination, and we would take

$$\mathit{Halt} = \mu X.\, \langle \mathtt{halt} \rangle\, \mathbf{true} \;\vee \bigwedge_{a \in Act \setminus \{\mathtt{halt}\}} [a]\, X$$

which expresses that M eventually performs halt.

However, VBPPs are much less powerful than Turing Machines. The idea of the reduction is to construct a VBPP which simulates the counter machine in a weak sense: the VBPP may execute many runs, some of which – the 'honest' runs – simulate the computation of the counter machine, while the rest are 'dishonest' runs in which, for instance, a counter is decreased by 2 instead of by 1. We replace the formula *Halt* above by a more complicated formula expressing that a certain honest run eventually executes halt. Thus, the problem of having a 'weaker' simulation is compensated by a 'stronger' formula, and it is still the case that the counter machine halts if and only if the BPP satisfies the new formula.

A 'weak' model of a counter machine. A counter c_j containing the number n is modeled by n copies in parallel of a process $\mathtt{C_j}$.

$$\mathtt{C_j} \stackrel{\mathrm{def}}{=} \mathtt{dec_j} \cdot \mathbf{0}$$

The action $\mathtt{dec_j}$ models decreasing the counter c_j by 1. Notice that VBPPs cannot enforce synchronisation between the action $\mathtt{dec_j}$ and a change of state of the counter machine. In some sense, the formula *Halt* will be in charge of modelling these synchronisations.

The states of the counter machine are modelled according to their associated instruction. A state q_i of type I is modelled by the processes

$$\begin{aligned} \mathtt{SQ_i} &\stackrel{\mathrm{def}}{=} \mathtt{in_i} \cdot (\mathtt{SQ_i} \parallel \mathtt{Q_i}) \\ \mathtt{Q_i} &\stackrel{\mathrm{def}}{=} \mathtt{out_i} \cdot (\mathtt{Q_k} \parallel \mathtt{C_j}) \end{aligned}$$

A state q_i of type II is modelled by the processes

$$\begin{aligned} \mathtt{SQ_i} &\stackrel{\mathrm{def}}{=} \mathtt{in_i} \cdot (\mathtt{SQ_i} \parallel \mathtt{Q_i}) \\ \mathtt{Q_i} &\stackrel{\mathrm{def}}{=} \mathtt{out_i} \cdot \mathbf{0} \end{aligned}$$

The actions $\mathtt{in_i}$ and $\mathtt{out_i}$ model the fact that the counter machine enters and leaves the state q_i, respectively. Notice that VBPPs cannot model the fact that from state q_i the states q_k or q'_k can be reached, because in order to model the choice between q_k and q'_k we need the choice operator.

The halting state q_{n+1} is modelled by the processes

$$\begin{aligned}\mathtt{SQ_{n+1}} &\overset{\text{def}}{=} \mathtt{in_{n+1}} \cdot (\mathtt{SQ_{n+1}} \parallel \mathtt{Q_{n+1}}) \\ \mathtt{Q_{n+1}} &\overset{\text{def}}{=} \mathtt{halt} \cdot \mathtt{Q_{n+1}}\end{aligned}$$

Finally, we define M as follows

$$\begin{aligned}\mathtt{SM} &\overset{\text{def}}{=} \mathtt{SQ_1} \parallel \ldots \parallel \mathtt{SQ_{n+1}} \\ \mathtt{M} &\overset{\text{def}}{=} \mathtt{SM} \parallel \mathtt{Q_0}\end{aligned}$$

It follows easily from the operational semantics of BPPs that every reachable state of M is strongly bisimilar to a unique expression of the form

$$\mathtt{SM} \parallel \mathtt{Q_0}^{i_0} \parallel \ldots \parallel \mathtt{Q_{n+1}}^{i_{n+1}} \parallel \mathtt{C_1}^{j_1} \parallel \ldots \parallel \mathtt{C_m}^{j_m}$$

where $\mathtt{P}^k \overset{\text{def}}{=} \underbrace{\mathtt{P} \parallel \ldots \parallel \mathtt{P}}_{k}$ for $k \geq 1$, and $\mathtt{P}^0 \overset{\text{def}}{=} \mathbf{0}$. For instance, the initial state M is strongly bisimilar to $\mathtt{SM} \parallel \mathtt{Q_0} \parallel \underbrace{\mathbf{0} \parallel \ldots \parallel \mathbf{0}}_{m+n-1}$. The expressions in which all the indices $i_0, \ldots, i_{n+1}$ except one, say i_j, are 0, and moreover $i_j = 1$, correspond to the configurations of the counter machine. The nonzero index indicates which is the state of the machine, and the indices $j_1, \ldots, j_m$ indicate the values of the counters. We say that the states which are bisimlar to a expression of this form are *meaningful*.

Let f be the mapping that assigns to each meaningful state the corresponding configuration of the counter machine. A run $\mathtt{M} \xrightarrow{a_1} E_1 \xrightarrow{a_2} E_2 \xrightarrow{a_3} \cdots$ is *honest* if it has a prefix satisfying the following property: the image under f of the subsequence of meaningful states of the prefix yields the computation of the counter machine $\mathcal{M}$. Loosely speaking, a honest run of M simulates the computation of the counter machine $\mathcal{M}$, but if the simulation terminates (with the execution of halt) then it may behave arbitrarily. It is clear from the definitions that M has honest runs, but not every run of M is honest.

We now construct a formula of the mu-calculus expressing that a certain honest run eventually executes halt. For a state q_i of type I we define the open formula

$$\phi_i = \langle \mathtt{out_i} \rangle \, Z$$

For a state q_i of type II we define

$$\begin{aligned}\phi_i = \; & [\mathtt{dec_j}] \, \mathbf{false} \;\wedge\; \langle \mathtt{out_i} \rangle \langle \mathtt{in_k} \rangle \, Z \\ & \vee \\ & \langle \mathtt{dec_j} \rangle \langle \mathtt{out_i} \rangle \langle \mathtt{in_{k'}} \rangle \, Z\end{aligned}$$

This formula expresses that either the action $\mathtt{dec_j}$ is not enabled (which models that the counter c_j is empty) and then after the execution of the sequence $\mathtt{out_i}$ $\mathtt{in_k}$, (which models the transition from the state q_i to the state q_k) the formula Z holds, or after the execution of the sequence $\mathtt{dec_j}$ $\mathtt{out_i}$ $\mathtt{in_{k'}}$ (which models decreasing counter c_j by 1 and moving from state q_i to q'_k) the formula Z holds.

Finally, we define

$$\mathit{Halt} = \mu Z. \, \langle \mathtt{halt} \rangle \, \mathbf{true} \;\vee\; \bigvee_{i=0}^{n} \phi_i$$

Loosely speaking, a run of `M` satisfies this formula if and only if it eventually executes the action `halt`, and the prefix preceeding the first occurrence of `halt` is the concatenation of small sequences of actions, each of which satisfies one of the formulas ϕ_i. Each of these small sequences simulates one step of the counter machine.

Theorem 4.1

The model checking problem for closed formulas of the modal mu-calculus and VBPPs is undecidable.

Proof: Let $\mathcal{M}$ be a counter machine with initially empty counters. The VBPP `M` satisfies the formula *Halt* iff it has a honest run that eventually executes the action `halt`. Since honest runs faithfully simulate the computation of $\mathcal{M}$, this is the case iff $\mathcal{M}$ halts. ■

Theorem 4.1 may not seem very surprising; the modal mu-calculus is known to be an extremely powerful logic, and one may expect it to be undecidable even for such simple processes as VBPPs. However, we will now show that the undecidability result still holds for a small subset of the modal mu-calculus.

4.3 Unified System of Branching Time

The Unified System of Branching Time (UB) is a logic introduced by Ben-Ari, Manna and Pnueli [2], where it is given a state-oriented semantics in terms of Kripke structures. In particular, UB includes a next operator $E\phi$, meaning that ϕ holds at some successor of the current state. Since BPPs are an action oriented model, we use a version of UB in which the next operator is relativised: for each action a, we have an operator $E(a)\phi$, whose meaning is that ϕ holds at some successor reached after performing an a. Also, we only allow the atomic sentence **true**, which holds at every state.

After these changes, the syntax of UB is

$$\phi ::= \mathbf{true} \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid E(a)\phi \mid EF\phi \mid EG\phi$$

UB is a fragment of Computation Tree Logic [10], which in turn can be easily embedded in the modal mu-calculus (see, for instance, [4]).

We interpret UB formulas on a rooted labelled transition system. We denote that a state s satisfies a formula ϕ by $s \models \phi$. The satisfaction relation is inductively defined as follows:

$s \models \mathbf{true}$		always
$s \models \neg\phi$	iff	not $s \models \phi$
$s \models \phi_1 \wedge \phi_2$	iff	$s \models \phi_1$ and $s \models \phi_2$
$s \models E(a)\phi$	iff	$s \xrightarrow{a} s'$ and $s' \models \phi$ for some s' such that
$s \models EF\phi$	iff	for some path $(s_1, s_2, \ldots)$ where $s = s_1$, $\exists i \geq 1\ s_i \models \phi$
$s \models EG\phi$	iff	for some path $(s_1, s_2, \ldots)$ where $s = s_1$, $\forall j \geq 1\ s_j \models \phi$

We define the following dual operators:

$$A(a) = \neg E(a)\neg \qquad AG = \neg EF\neg \qquad AF = \neg EG\neg$$

A rooted transition system satisfies a formula ϕ if the initial state satisfies ϕ. The model checking problem for a formula ϕ and a BPP or a Petri net consists in deciding if the initial state of the associated transition system satisfies the formula.

In the rest of the section we consider two different fragments of UB, which we denote by EF and EG. The syntax of EF is obtained from the syntax of UB by removing the operator EG (and implicitely its dual AF). So EF can only express properties of the form 'always in the future ϕ holds' ($AG\phi$) or 'sometime in the future ϕ holds' ($EF\phi$), but not properties like 'eventually in the future ϕ holds' ($AF\phi$). In [12] this fragment is called UB$^-$. The syntax of EG is obtained from the syntax of UB by removing the operator EF (and implicitely its dual AG).

We prove the following three results:

- The model checking problem for EG and VBPPs is undecidable.
- The model checking problem for EF and Petri nets is undecidable.
- The model checking problem for EF and BPPs is decidable and PSPACE-hard. Moreover, the model checking problem for EF is decidable for any class of Petri nets whose set of reachable markings is effectively semilinear (a concept to be defined).

4.4 Undecidability of EG for VBPPs

We proceed again by a reduction from the halting problem for counter machines. Given a counter machine $\mathcal{M}$, we construct a VBPP weak model of `M` and a formula ϕ_h of EG satisfying the following two properties:

(1) there exists a run of `M` such that all the states reached along it satisfy ϕ_h, and

(2) if all the states reached along a run of `M` satisfy ϕ_h, then the run is honest.

Let us see that, once `M` and ϕ_h have been given, the undecidability result follows easily. We define the formula

$$Halt = AF(\ \neg\phi_h\ \vee\ E(\mathtt{halt})\,\mathbf{true}\ )$$

If `M` satisfies *Halt*, then the runs that satisfy ϕ_h at every state must contain a state satisfying $E(\mathtt{halt})\,\mathbf{true}$. Since such runs exist and are honest by (1) and (2), and since honest runs faithfully simulate the behaviour of the counter machine, the counter machine terminates.

Conversely, assume that the counter machine terminates. A run of the model `M` either is honest or visits a state which does not satisfy ϕ_h. In the first case, since the machine terminates, the run contains a state satisfying $E(\mathtt{halt})\,\mathbf{true}$, and therefore it satisfies *Halt*. In the second case, the run directly satisfies *Halt*.

Unfortunately, it is not possible to find the formula ϕ_h for the weak model of the counter machine we use for the modal mu-calculus. The essence of the problem is easier to see in a simpler process `P` given by

$$\mathtt{P} \overset{\text{def}}{=} \mathtt{Q} \,||\, \mathtt{R} \qquad\qquad \mathtt{Q} \overset{\text{def}}{=} \mathtt{q}\cdot\mathtt{Q} \qquad\qquad \mathtt{R} \overset{\text{def}}{=} \mathtt{r}\cdot\mathtt{R}$$

Assume that `P` is simulating, in a weak sense, a certain system. Assume further that the honest runs of `P` are those whose prefixes satisfy $|\#\mathtt{q} - \#\mathtt{r}| \leq 1$, where $\#\mathtt{q}$, $\#\mathtt{r}$ is the number of `q`'s and `r`'s in the prefix. It is easy to see that no formula ϕ_h of UB satisfies the conditions (1) and (2) above. The reason is that, for every run $E_0 \xrightarrow{\mathtt{a}_1} E_1 \xrightarrow{\mathtt{a}_2} \ldots$ of `P`, we have $\mathtt{P} = E_0 = E_1 = \ldots$. Therefore, all intermediate states satisfy exactly the same properties, and no formula of EG (or UB) can distinguish between honest and dishonest runs.

The rest of the section is devoted to finding another weak model of a counter machine for which the formula ϕ_h exists.

We shall often use formulas of the form $E(a)\,\mathbf{true}$. To lighten the notation, we define

$$EN(a) = E(a)\,\mathbf{true}$$

$EN(a)$ is read 'a is enabled'.

The solution. We make use of an idea introduced by Yoram Hirshfeld in [18]. If we are allowed to split the actions `q` and `r` in the definition of the process `P` above, things change. Define

$$\mathtt{P} \stackrel{\text{def}}{=} \mathtt{Q} \parallel \mathtt{R} \qquad \mathtt{Q} \stackrel{\text{def}}{=} \mathtt{q}^1 \cdot \mathtt{q}^2 \cdot \mathtt{q}^3 \cdot \mathtt{Q} \qquad \mathtt{R} \stackrel{\text{def}}{=} \mathtt{r}^1 \cdot \mathtt{r}^2 \cdot \mathtt{r}^3 \cdot \mathtt{R}$$

and define the honest runs as those satisfying $|\#\mathtt{q}^1 - \#\mathtt{r}^1| \leq 1$. The following formula ϕ_h holds for all the states of a run only if it performs the sequence $(\mathtt{q}^1\ \mathtt{r}^1\ \mathtt{q}^2\ \mathtt{r}^2\ \mathtt{q}^3\ \mathtt{r}^3)^\omega$, which is honest.

$$\begin{array}{rcllll} \phi_h & = & (EN(\mathtt{q}^1) \wedge EN(\mathtt{r}^1)) & \vee & (EN(\mathtt{q}^2) \wedge EN(\mathtt{r}^1)) & \vee \\ & & (EN(\mathtt{q}^2) \wedge EN(\mathtt{r}^2)) & \vee & (EN(\mathtt{q}^3) \wedge EN(\mathtt{r}^2)) & \vee \\ & & (EN(\mathtt{q}^3) \wedge EN(\mathtt{r}^3)) & \vee & (EN(\mathtt{q}^1) \wedge EN(\mathtt{r}^3)) & \end{array}$$

From `P`, only the action $\mathtt{q}^1$ can occur, because if $\mathtt{r}^1$ occurs, then a state is reached which enables $\mathtt{q}^1$ and $\mathtt{r}^2$, and such a state does not satisfy ϕ_h. We can then similarly prove that after $\mathtt{q}^1$ only $\mathtt{r}^1$ can occur, and so on. Incidentally, the reader can check that splitting `p` and `q` into two is not enough.

Once the power of splitting has been identified, the rest is straightforward. We refine our VBPP model in the following way.

A second 'weak' model. A counter c_j is now modelled by

$$\mathtt{C_j} \stackrel{\text{def}}{=} \mathtt{dec}_\mathtt{j}^1 \cdot \mathtt{dec}_\mathtt{j}^2 \cdot \mathtt{dec}_\mathtt{j}^3 \cdot \mathbf{0}$$

A state q_i of type II is modelled by

$$\begin{array}{rcl} \mathtt{SQ_i} & \stackrel{\text{def}}{=} & \mathtt{in_i} \cdot (\mathtt{Q_i} \parallel \mathtt{SQ_i}) \\ \mathtt{Q_i} & \stackrel{\text{def}}{=} & \mathtt{out}_\mathtt{i}^1 \cdot \mathtt{out}_\mathtt{i}^2 \cdot \mathtt{out}_\mathtt{i}^3 \cdot \mathbf{0} \end{array}$$

In the other equations we replace $\mathtt{out_i}$ by $\mathtt{out}_\mathtt{i}^1$ for consistency, but the actions $\mathtt{out_i}$ are not split.

For the definition of the formula ϕ_h is convenient to introduce some notations. First of all, in order to allow a more compact representation of the formula, we rename the action `halt` to $\mathtt{out}_\mathtt{n+1}^1$. Now, define

$$A = \{\mathtt{out}_\mathtt{i}^1, \mathtt{out}_\mathtt{i}^2, \mathtt{out}_\mathtt{i}^3 \mid 0 \leq i \leq n+1\} \cup \{\mathtt{dec}_\mathtt{j}^2, \mathtt{dec}_\mathtt{j}^3 \mid 1 \leq j \leq m\}$$

(notice that the actions $\mathtt{dec}_\mathtt{i}^1$ do not belong to A), and let $a_1, \ldots, a_k$ be actions of A. Then,

$$\widehat{EN}(a_1, \ldots, a_k) \;=\; \bigwedge_{i=1}^{k} EN(a_i) \;\wedge\; \bigwedge_{i=1}^{k} \neg E(a_i)EN(a_i) \;\wedge \bigwedge_{a \in A \setminus \{a_1 \ldots a_k\}} \neg EN(a)$$

In words, $\widehat{EN}(a_1,\ldots,a_k)$ states that the actions $a_1,\ldots a_k$ are enabled, no sequence $a_i\ a_i$ is enabled, and all the other actions of A are disabled.

The formula ϕ_h is a disjunction of formulas. For each state q_i of type I and for the state q_{n+1}, ϕ_h contains a disjunct of the form

$$\widehat{EN}(\mathtt{out}_{\mathtt{i}}^{1})$$

For each state q_i of type II, ϕ_h contains two disjuncts. The first is

$$\begin{aligned}\neg EN(\mathtt{dec}_{\mathtt{j}}^{1}) \quad \wedge \quad (\quad & \widehat{EN}(\mathtt{out}_{\mathtt{i}}^{1}) \vee \\ & \widehat{EN}(\mathtt{out}_{\mathtt{i}}^{2}) \vee \\ & \widehat{EN}(\mathtt{out}_{\mathtt{i}}^{2}, \mathtt{out}_{\mathtt{k}}^{1}) \vee \\ & \widehat{EN}(\mathtt{out}_{\mathtt{i}}^{3}, \mathtt{out}_{\mathtt{k}}^{1}) \qquad)\end{aligned}$$

and the second is

$$\begin{aligned}&(\ EN(\mathtt{dec}_{\mathtt{j}}^{1}) \wedge \widehat{EN}(\mathtt{out}_{\mathtt{i}}^{1}) \ ) \vee \\ &(\ EN(\mathtt{dec}_{\mathtt{j}}^{1}) \wedge \widehat{EN}(\mathtt{out}_{\mathtt{i}}^{2}) \ ) \vee \\ &(\ EN(\mathtt{dec}_{\mathtt{j}}^{1}) \wedge \widehat{EN}(\mathtt{out}_{\mathtt{i}}^{2}, \mathtt{out}_{\mathtt{k'}}^{1}) \ ) \vee \\ &\widehat{EN}(\mathtt{dec}_{\mathtt{j}}^{2}, \mathtt{out}_{\mathtt{i}}^{2}, \mathtt{out}_{\mathtt{k'}}^{1}) \vee \\ &\widehat{EN}(\mathtt{dec}_{\mathtt{j}}^{2}, \mathtt{out}_{\mathtt{i}}^{3}, \mathtt{out}_{\mathtt{k'}}^{1}) \vee \\ &\widehat{EN}(\mathtt{dec}_{\mathtt{j}}^{3}, \mathtt{out}_{\mathtt{i}}^{3}, \mathtt{out}_{\mathtt{k'}}^{1}) \vee \\ &\widehat{EN}(\mathtt{dec}_{\mathtt{j}}^{3}, \mathtt{out}_{\mathtt{k'}}^{1}) \qquad\qquad)\end{aligned}$$

It is not difficult to see that there exists a run of M such that all the states it visits satisfy ϕ_h. This run simulates the computation of the counter machine and, in case it terminates, executes the sequence $\mathtt{halt}^{\omega} = \mathtt{out_{n+1}}^{\omega}$. We now prove:

Lemma 4.2

If all the states of a run of the model M *satisfy the formula ϕ_h, then the run is honest.*

Proof: Let

$$E \sim \mathtt{SM} \parallel \mathtt{Q_i} \parallel \mathtt{C_1}^{i_1} \parallel \ldots \parallel \mathtt{C_n}^{i_n}$$

be a meaningful state of a run in which every state satisfies ϕ_h, and assume $i \neq n+1$, i.e., q_i is not the halting state. We show that the next meaningful state of the run is the one that corresponds to the next configuration in the computation of the counter machine. More concretely, we examine the actions enabled at E, and check that only one leads to a state E' satisfying ϕ_h. Then we examine the actions enabled at E', check again that only one leads to a state satisfying ϕ_h, and so on. We terminate when a sequence of actions leading to a meaningful state has been determined.

Let c_j be the counter corresponding to the state q_i that appears in E. There are three possible cases: (1) q_i is of type I; (2) q_i is of type II, and $i_j = 0$; (3) q_i is of type

II, and $i_j > 0$. Since the procedure is very repetitive and requires to consider many different cases, we only deal in detail with the case (2), i.e., the case in which q_i is of the form

$$\text{if } c_j = 0 \text{ then goto } q_k \text{ else } (c_j := c_j - 1; \text{ goto } q_{k'})$$

for some j, k, k', and E contains no copies of the process C_j, i.e. we have $i_j = 0$.

The actions enabled at E are $\mathtt{out}^1_\mathtt{i}$, all the $\mathtt{in}$ actions, and the $\mathtt{dec}^1$ actions of the counters which are nonempty at E (in particular, since $i_j = 0$, the action $\mathtt{dec}^1_\mathtt{j}$ is not enabled). For each state F reached by the occurrence of one of these actions, we exhibit either a disjunct of ϕ_h satisfied by F (items marked with $\bullet$), or a feature of F from which it is easy to deduce that F does not satisfy any disjunct of ϕ_h (items marked with $-$).

- The $\mathtt{in}$ actions lead to states which enable $\mathtt{out}^1_\mathtt{i}$ and some other $\mathtt{out}^1$ action, or the sequence $\mathtt{out}^1_\mathtt{i}\ \mathtt{out}^1_\mathtt{i}$.
- The $\mathtt{dec}^1$ actions lead to states which enable $\mathtt{out}^1$ and one $\mathtt{dec}^2$ action, but no other action of A.
- $\bullet$ The action $\mathtt{out}^1_\mathtt{i}$ leads to the state

$$E_1 \sim \mathtt{SM} \parallel \mathtt{out}^2_\mathtt{i} \cdot \mathbf{0} \parallel \mathtt{C_1}^{i_1} \parallel \ldots \parallel \mathtt{C_n}^{i_n}$$

satisfying $\neg EN(\mathtt{dec}^1_\mathtt{j}) \wedge \widehat{EN}(\mathtt{out}^2_\mathtt{i})$.

The actions enabled at E_1 are $\mathtt{out}^2_\mathtt{i}$, all the $\mathtt{in}$ actions, and the $\mathtt{dec}^1$ actions of the nonempty counters.

- The action $\mathtt{out}^2_\mathtt{i}$ leads to a state which enables no $\mathtt{out}$ actions.
- The $\mathtt{dec}^1$ actions lead to states which enable one $\mathtt{out}^2$ and one $\mathtt{dec}^2$ action, but no other action of A.
- The actions $\mathtt{in_l}$, $l \neq k$, lead to states which enable $\mathtt{out}^2_\mathtt{i}$ and $\mathtt{out}^1_\mathtt{l}$, but neither $\mathtt{dec}^1_\mathtt{j}$ nor any other action of A.
- $\bullet$ The action $\mathtt{in_k}$ leads to the state

$$E_2 \sim \mathtt{SM} \parallel \mathtt{out}^2_\mathtt{i} \cdot \mathbf{0} \parallel \mathtt{Q_k} \parallel \mathtt{C_1}^{i_1} \parallel \ldots \parallel \mathtt{C_n}^{i_n}$$

satisfying $\neg EN(\mathtt{dec}^1_\mathtt{j}) \wedge \widehat{EN}(\mathtt{out}^2_\mathtt{i}, \mathtt{out}^1_\mathtt{k})$.

The actions enabled at E_2 are $\mathtt{out}^1_\mathtt{k}$, $\mathtt{out}^2_\mathtt{i}$, all the $\mathtt{in}$ actions, and the $\mathtt{dec}^1$ actions of the nonempty counters.

- The action $\mathtt{out}^1_\mathtt{k}$ leads to a state which enables $\mathtt{out}^2_\mathtt{i}$ and $\mathtt{out}^2_\mathtt{k}$.
- The $\mathtt{in}$ actions lead to states which enable either three different $\mathtt{out}$ actions or the sequence $\mathtt{out}^1_\mathtt{k}\mathtt{out}^1_\mathtt{k}$.
- The $\mathtt{dec}^1$ actions lead to states which enable $\mathtt{out}^2_\mathtt{i}$, $\mathtt{out}^1_\mathtt{k'}$ and one $\mathtt{dec}^2$ action different from $\mathtt{dec_j}^2$.
- $\bullet$ The action $\mathtt{out}^2_\mathtt{i}$ leads to the state

$$E_3 \sim \mathtt{SM} \parallel \mathtt{out}^3_\mathtt{i} \cdot \mathbf{0} \parallel \mathtt{Q_k} \parallel \mathtt{C_1}^{i_1} \parallel \ldots \parallel \mathtt{C_n}^{i_n}$$

satisfying $\neg EN(\mathtt{dec}^1_\mathtt{j}) \wedge \widehat{EN}(\mathtt{out}^3_\mathtt{i}, \mathtt{out}^1_\mathtt{k})$.

The actions enabled at E_3 are $\mathtt{out}_\mathtt{k}^1$, $\mathtt{out}_\mathtt{i}^3$, all the `in` actions, and the $\mathtt{dec}^1$ actions of the nonempty counters.

- The action $\mathtt{out}_\mathtt{k}^1$ leads to a state which enables $\mathtt{out}_\mathtt{i}^3$ and $\mathtt{out}_\mathtt{k}^2$.
- The `in` actions lead to states which enable either three different `out` actions or the sequence $\mathtt{out}_\mathtt{k}^1\mathtt{out}_\mathtt{k}^1$.
- The $\mathtt{dec}^1$ actions lead to states which enable $\mathtt{out}_\mathtt{i}^3$, $\mathtt{out}_\mathtt{k'}^1$ and one $\mathtt{dec}^2$ action different from ${dec_j}^2$ are enabled.

- The action $\mathtt{out}_\mathtt{i}^3$ leads to the state

$$E_4 \sim \mathtt{SM} \parallel \mathtt{Q_k} \parallel \mathtt{C_1}^{i_1} \parallel \ldots \parallel \mathtt{C_n}^{i_n}$$

satisfying $\neg EN(\mathtt{dec}_\mathtt{j}^1) \wedge \widehat{EN}(\mathtt{out}_\mathtt{i}^3, \mathtt{out}_\mathtt{k}^1)$.

E_4 is a meaningful state. Therefore, the run executes $\mathtt{out}_\mathtt{i}^1\ \mathtt{in}_\mathtt{k}^1\ \mathtt{out}_\mathtt{i}^2\ \mathtt{out}_\mathtt{i}^3$ from E. This sequence faithfully simulates the transition from q_i to q_k while keeping the counter c_j to 0.

This concludes the discussion of case (2). In cases (1) and (3) we proceed similarly. The only possible sequence in case (1) is $\mathtt{out}_\mathtt{i}^1$. The only possible sequence in case (3) is

$$\mathtt{out}_\mathtt{i}^1\ \mathtt{out}_\mathtt{k'}^1\ \mathtt{dec}_\mathtt{j}^1\ \mathtt{out}_\mathtt{i}^2\ \mathtt{dec}_\mathtt{j}^2\ \mathtt{out}_\mathtt{i}^2\ \mathtt{dec}_\mathtt{j}^3$$

Again, these sequences faithfully simulate the computation of the counter machine.

■

Theorem 4.3

The model checking problem for EG and VBPPs is undecidable.

Proof: Use the argument presented at the beginning of the section to prove that a machine $\mathcal{M}$ terminates iff the model `M` satisfies the formula *Halt*. ■

4.5 Undecidability of EF for arbitrary Petri nets.

This result was obtained in [15]. Here we briefly sketch the proof.

We prove undecidability by a reduction from the *containment problem*. An instance of the problem is a triple (Σ_1, Σ_2, f), where Σ_1, Σ_2 are two Petri nets having the same number of places, and f is a bijection between the sets of places of Σ_1 and Σ_2. The mapping f can be extended to a bijection between markings in the obvious way. The question to decide is whether for every reachable marking M of Σ_1, $f(M)$ is a reachable marking of Σ_2. Rabin showed that this problem is undecidable. A proof can be found in [24].

A look at this proof shows that one can safely assume that all the transitions of Σ_1 and Σ_2 have some input place, and that all their arcs have weight 1. Let (Σ_1, Σ_2, f) be an instance of this restricted containment problem, and let T_1, T_2 be the sets of transtions of Σ_1, Σ_2. Assume without loss of generality that the sets of nodes of Σ_1 and Σ_2 are disjoint. We construct an unlabelled Petri net Σ. The description of Σ given in [15] has a mistake which is corrected here.

The construction takes place in several steps. At each step we add new places, transitions, and arcs. We represent a node n (place or transition) having X and Y as input and output sets of nodes as $X \rightarrow Y$. If we wish to give a name n to the node, we write $n: X \rightarrow Y$. All the arcs of Σ are going to have weight 1, and therefore with this notation we specify at the same time the set of nodes and the weighting function. The steps are:

- Add Σ_1 and Σ_2.
- Add three places $A : T_1 \rightarrow T_1$, $B : T_2 \rightarrow T_2$, $C : \emptyset \rightarrow \emptyset$.
- Add two transitions $t_{AB}: \{A\} \rightarrow \{B\}$, $t_{BC}: \{B\} \rightarrow \{C\}$.
  Put one token on A, and no tokens on B and C.
- For every place s of Σ_1, add a transition $\{s, f(s), C\} \rightarrow \{C\}$.
- For every place s added so far, except C, add a transition $\{s\} \rightarrow \{s\}$.

Σ simulates Σ_1 until transition t_{AB} occurs. Then, no transition of Σ_1 can occur anymore, and Σ simulates Σ_2 until transition t_{BC} occurs. After that, pairs of tokens may be extracted simultaneously from a place s and its image $f(s)$.

Let *Dead* be the conjunction of the formulas $\neg EN(t)$ for every transition t of Σ. A marking satisfies *Dead* iff it is dead, i.e., it does not enable any transition. It is proved in [15] that a triple (Σ_1, Σ_2, f) is a 'yes' instance of the containment problem iff Σ satisfies the formula

$$AG(\ EN(t_{AB}) \implies E(t_{AB})\ EF\ \mathit{Dead}\ )$$

This formula states that every sequence of the form $M_0 \xrightarrow{\sigma_1} M_1 \xrightarrow{t_{AB}} M_2$ can be extended to a sequence $M_0 \xrightarrow{\sigma_1} M_1 \xrightarrow{t_{AB}} M_2 \xrightarrow{\tau} M_3$ where M_3 is a dead marking. Since every place s except C has a transition $\{s\} \rightarrow \{s\}$, the only dead reachable marking of Σ is the one which puts one token on C and no tokens elsewhere. Therefore, τ is a sequence which first generates the marking $f(M_1)$ in Σ_2, and then after the occurrence of t_{BC} 'empties' both Σ_1 and Σ_2. So, if the formula holds, for every reachable marking M of Σ_1, $f(M)$ is a reachable marking of Σ_2. The converse direction is proved similarly.

4.6 Decidability of EF for BPPs and other Petri net classes

We prove the decidability of the model checking problem for EF and Petri nets whose reachability relation is effectively semilinear. We first introduce this notion.

The markings of a Petri net with k places can be represented as vectors of k natural numbers: the i-th component of the vector indicates the number of tokens on the i-th place, for some fixed order of the places. We can then identify the set of reachable markings with a set of vectors.

A set X of elements of $\mathbb{N}^k$ is *linear* if there exist $v, v_1, \ldots, v_n \in \mathbb{N}^k$ such that

$$X = \{v + a_1 v_1 + \ldots + a_n v_n \mid a_1, \ldots, a_n \geq 0\}$$

X is *semilinear* if it is the union of a finite number of linear sets.

It is easy to see that every semilinear set can be expressed in Presburger arithmetic, the first order theory of addition (see for instance [3]). That is, given a semilinear set $X \subseteq \mathbb{N}^k$, there exists a formula of Presburger arithmetic with k free integer variables $\mathbf{x_1}, \ldots, \mathbf{x_k}$ which holds for an assignment $\mathbf{x_1} := n_1, \ldots, \mathbf{x_k} := n_k$ if and only if $(n_1, \ldots, n_k) \in X$.

The reachability relation of a net N is the set of pairs (M, M') of markings of N such that M' is reachable from M. When the reachability relation of N is semilinear, we denote by $\mathit{Reach}_N(\mathbf{M}, \mathbf{M'})$ the formula of Presburger arithmetic expressing it ($\mathbf{M}$ and $\mathbf{M'}$ are two vectors of free variables). The reachability relations of a class of Petri nets are *effectively semilinear* if they are semilinear sets, and moreover there is an algorithm which given a net N of the class computes $\mathit{Reach}_N(\mathbf{M}, \mathbf{M'})$.

To prove decidability, we reduce the problem to the problem of determining if a given formula of Presburger arithmetic is true, which is decidable ([3]).

Theorem 4.4

Let $\mathcal{C}$ be a class of nets whose reachability relations are effectively semilinear. The model checking problem for EF and $\mathcal{C}$ is decidable.

Proof: Let $\Sigma = (N, M_0)$ be a Petri net of $\mathcal{C}$, and let ϕ be a formula of of EF. We prove that the set of reachable markings that satisfy ϕ can be expressed by a formula $F_\phi(\mathbf{M})$ of Presburger arithmetic, which is moreover effectively computable.

The proof is by structural induction on the syntax of EF:

- $\phi = \mathbf{true}$. The set of reachable markings that satisfy ϕ is the set of all reachable markings of Σ, which is effectively semilinear by hypothesis. Take $F_{\mathbf{true}}(\mathbf{M}) = \mathit{Reach}_N(M_0, \mathbf{M})$ (i.e., we assign M_0 to $\mathbf{M}$).
- $\phi = \neg\psi$. Then $F_\phi(\mathbf{M}) = \neg F_\psi(\mathbf{M})$.
- $\phi = \phi_1 \wedge \phi_2$. Then $F_\phi(\mathbf{M}) = F_{\phi_1}(\mathbf{M}) \wedge F_{\phi_2}(\mathbf{M})$.
- $\phi = E(a)\psi$. A marking satisfies $E(a)\psi$ if it enables some transition with label a and the marking reached after executing this transition satisfies ψ. All this can be easily encoded in Presburger arithmetic. For instance, let $s_1, \ldots, s_k$ be the places of Σ, and assume that the input places of a transition t are s_1 and s_2 and its output places s_3 and s_4. Let $\mathbf{M} = (\mathbf{m_1}, \ldots, \mathbf{m_k})$ be the set of variables corresponding to the places $s_1, \ldots, s_k$. Then the formula

$$\mathbf{m_1} > 0 \wedge \mathbf{m_2} > 0 \wedge F_\psi(\mathbf{m_1} - 1, \mathbf{m_2} - 1, \mathbf{m_3} + 1, \mathbf{m_4} + 1, \mathbf{m_5}, \ldots, \mathbf{m_k})$$

  encodes the markings which enable t and, once t occurs, are transformed into markings satisfying ψ.
- $\phi = EF\psi$. Then $F_{EF\psi}(\mathbf{M}) = \exists \mathbf{M'}\ (\ \mathit{Reach}_N(\mathbf{M}, \mathbf{M'})\ \wedge\ F_\psi(\mathbf{M'})\ )$.

Now, to check if Σ satisfies a formula ϕ, it suffices to decide if the formula $F_\psi(M_0)$ is true. ■

It was proved in [14] that the reachability relations of the Petri nets corresponding to BPPs in normal form are effectively semilinear (this property also holds for persistent and reversible Petri nets [16]). In this particular case, we can obtain the following lower bound:

Theorem 4.5

The model checking problem for EF and BPPs is PSPACE-hard, even if the BPPs are finite state.

Proof: The proof is a straightforward reduction from the problem of deciding if a quantified boolean formula without free variables evaluates to 'true', which is known to be PSPACE-complete [1]. Consider without loss of generality a quantified boolean formula with alternating quantifiers

$$\Phi = \exists x_1 \forall x_2 \exists x_3 \cdots \exists x_n \, \phi(x_1 \cdots x_n)$$

where $\phi(x_1 \cdots x_n)$ is a boolean formula in conjunctive normal form. We construct in polynomial space a BPP with leading variable S and a formula *True_Φ* of EF such that Φ is true iff S satisfies *True_Φ*.

Let $\phi(x_1, \ldots, x_n) = C_1 \wedge C_2 \wedge \ldots \wedge C_m$, where each C_j is a clause. For each clause C_j, define the BPP

$$\mathtt{True\_C_j} \overset{\mathrm{def}}{=} \mathtt{true\_C_j} \cdot \mathbf{0}$$

For each variable x_i, let $\mathcal{C}_i^t$ be the set of clauses of ϕ which contain the literal x_i (i.e., the set of clauses that become true when X_i is set to true), and let $\mathcal{C}_i^f$ be the set of clauses that contain the literal $\neg x_i$ (i.e., the set of clauses that become true when X_i is set to false). Define

$$\begin{array}{rcl} \mathtt{X_i} & \overset{\mathrm{def}}{=} & \mathtt{assign\_x_i} \cdot \mathtt{True\_x_i} + \mathtt{assign\_x_i} \cdot \mathtt{False\_x_i} \\ \mathtt{True\_x_i} & \overset{\mathrm{def}}{=} & \mathtt{prop\_x_i} \cdot (\parallel_{C_j \in \mathcal{C}_i^t} \mathtt{True\_C_j}) \\ \mathtt{False\_x_i} & \overset{\mathrm{def}}{=} & \mathtt{prop\_x_i} \cdot (\parallel_{C_k \in \mathcal{C}_i^f} \mathtt{True\_C_k}) \end{array}$$

The process $\mathtt{X_i}$ can perform an $\mathtt{assign\_x_i}$ action, which models setting x_i to true or to false, respectively. If x_i is set to true, then all the clauses that contain the literal x_i are true as well, otherwise the clauses that contain the literal $\neg x_i$ are true. This *propagation* of the truth value of x_i to the clauses that contain either x_i or $\neg x_i$ is modeled by the action $\mathtt{prop\_x_i}$.

Finally, define the leading variable of the BPP as follows:

$$\mathtt{S} \overset{\mathrm{def}}{=} \mathtt{X_1} \parallel \ldots \parallel \mathtt{X_n}$$

Since none of the equations of this BPP is recursive, S is a finite state process.

Let us now construct the formula *True_Φ*. First, we define for every $0 \leq i \leq n$ the following auxiliary formula

$$\begin{array}{rcl} \mathit{Assigned}[i] & = & EN(\mathtt{prop\_x_1}) \wedge \ldots \wedge EN(\mathtt{prop\_x_i}) \wedge \\ & & EN(\mathtt{assign\_x_{i+1}}) \wedge \ldots EN(\mathtt{assign\_x_n}) \end{array}$$

Loosely speaking, *Assigned*$[i]$ expresses the following:

(1) the variables x_1 to x_i (or no variable in the case $i = 0$) have been already set to true or false, but the values have not been propagated (otherwise the prop actions would no longer be enabled), and

(2) the variables $x_{i+1}, \ldots, x_n$ have not been assigned any truth value yet (otherwise the assign actions would no longer be enabled.

We define the formula *True_Φ* as follows:

$$\begin{array}{l}
EF(\mathit{Assigned}[1] \wedge \\
\quad AG(\mathit{Assigned}[2] \Rightarrow \\
\quad\quad EF(\mathit{Assigned}[3] \wedge \\
\quad\quad\quad \ldots \\
\quad\quad\quad\quad EF(\mathit{Assigned}[n] \wedge \\
\quad\quad\quad\quad\quad EN(\texttt{true\_C}_1) \wedge \ldots \wedge EN(\texttt{true\_C}_n)) \ldots)))
\end{array}$$

This formula expresses that there is a truth value for x_1 such that for every truth value for x_2 such that ...there exists a truth value for x_n such that every clause of the formula ϕ is true. So S satisfies this formula iff the formula Φ is true. ■

Using results of [14], it is easy to show that the model checking problem for formulas of EF without nested temporal operators (and arbitrary BPPs) is NP-complete. We conjecture that the problem for formulas with k nested temporal operators is complete for the k-th level of the polynomial time hierarchy, and that the model checking problem for arbitrary formulas is PSPACE-complete, which would yield an exponential time upper bound. Since it is known that checking validity in Presburger arithmetic requires at least doubly exponential nondeterministic time [17], this result would improve the upper bound derived from Proposition 4.4.

5 Adding atomic sentences

We study the impact on decidability caused by the addition of certain atomic sentences to the logics we have studied so far. By atomic sentences we understand propositional constants whose semantics is determined *a priori* by a valuation. In mu-calculi, atomic sentences are equivalent to free variables. In the logic UB described in the last section the only atomic sentence was **true**.

5.1 Linear Time logics

We add atomic sentences to a weak linear time temporal logic, and show that there exist simple valuations which make the model checking problem undecidable for VBPPs. More precisely, we show that, using these valuations, it is possible to construct a formula *Halt'* equivalent to the formula *Halt* of EF, i.e., true for the same transition systems.

The syntax of the logic is

$$\phi ::= \mathbf{true} \mid \mathbf{EN(a)} \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid (a)\phi \mid F\phi$$

where a is an action of Act, and $\mathbf{EN(a)}$ is an atomic sentence.

The logic is interpreted on the paths of a labelled transition system. We consider the valuation which assigns to $\mathbf{EN(a)}$ the paths which start at a state which enables a. Negation, conjunction and **true** are interpreted as usual. A path satisfies $F\phi$ if it contains a suffix which satisfies ϕ. A labelled transition system $\mathcal{T}$ with an initial state s_0 satisfies a formula ϕ if all the runs of $\mathcal{T}$ (i.e., all the paths of $\mathcal{T}$ that start at s_0) satisfy ϕ. In particular, a transition system satisfies $\mathbf{EN(a)}$ if and only it satisfies the formula $EN(a)$ of UB.

We define the formula ϕ'_h of this logic as the result of replacing each occurrence of an expression $EN(a)$ in the formula ϕ_h of EG by the sentence $\mathbf{EN(a)}$.

We now construct the formula

$$Halt' = F(\ \neg\phi'_h\ \vee\ \mathbf{EN}(\mathbf{halt})\ )$$

A transition system satisfies $Halt'$ if every run visits a state which satisfies $\neg\phi'_h\ \vee$ $\mathbf{EN}(\mathbf{halt})$. Clearly, this is the also the meaning of the formula $Halt = AF(\ \neg\phi_h\ \vee$ $EN(\texttt{halt}))$ of EG. It follows that the model checking problem for this linear temporal logic and VBPPs is undecidable.

Notice that the addition of the atomic sentences has a dramatic effect: without them, the model checking problem for closed formulas of the linear mu-calculus and arbitrary Petri nets is decidable; with them, the model checking problem for the weak temporal logic given above and VBPPs is undecidable.

When model checking Petri nets it is common to use logics with atomic sentences that express properties of the markings visited by a path. For instance, a sentence like $s = 0$ is assigned the set of paths starting at markings which put no tokens on the place s. It is easy to see that these sentences can 'simulate' the sentences $\mathbf{EN}(\mathbf{a})$, and so the undecidability results also hold for them.

5.2 Branching Time Logics

Theorem 4.4 has been proved in Section 4 for the logic EF, which has only **true** as atomic sentence. Actually, the decidability result still holds if we add atomic sentences, as long as we restrict ourselves to valuations which are expressible in Presburger arithmetic (i.e., valuations which assign to a given sentence a set of states expressible in Presburger arithmetic). In particular, we can add sentences of the form

$$A \cdot \mathbf{M} \leq B$$

where A is a matrix of integers, $\mathbf{M}$ a vector of marking variables, and B a vector of integers. An example of a formula of this extension is

$$AG\,(\ \mathbf{m_1} + \mathbf{m_2} = 3\ \implies\ EF\ \mathbf{m_3} > 5\ )$$

which expresses that every marking that puts a total of 3 tokens on the places s_1 and s_2 has a successor which puts more than 5 tokens on the place s_3.

In the case of a Petri net derived from BPPs, [14] not only proves the semilinearity of the reachability relation, but also the semilinearity of the set of *Parikh mappings* of the sequences of actions executed by the net. Given a sequence $\sigma \in Act^*$, its Parikh mapping $P(\sigma)$ assigns to each action the number of times it occurrs in σ. This result allows us to extend EF further while keeping decidability. For instance, it is possible to relativise not only the next operator, but also the other operators. Given an effectively semilinear set $\mathcal{P}$ of Parikh mappings, we can introduce an operator $EF(\mathcal{P})\phi$, which holds at a marking M if there exists a marking M' that satisfies ϕ, and moreover M' can be reached from M by the execution of a sequence of actions whose Parikh mapping belongs to $\mathcal{P}$. $AG(\mathcal{P})$ is defined similarly.

6 Conclusions

We have studied the model checking problem for Petri nets and Basic Parallel processes (BPPs), two models of concurrent computation, and different temporal logics. While

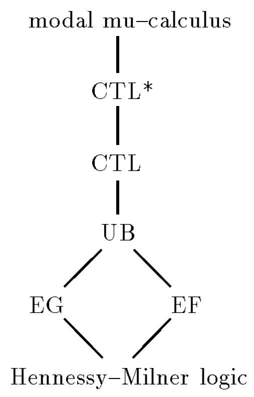

Fig. 2 Branching time logics

Petri nets are a rather powerful model, BPPs are very weak. We have also studied an even weaker model, the Very Basic Parallel Processes (VBPPs).

The following table summarizes our results. F+**EN** denotes the linear time logic having F as single modal operator and atomic sentences of the form **EN**(**a**) (see section 5.1). EF+**Pres** denotes the branching time logic having EF as modal operator and atomic sentences evaluated to sets expressible in Presburger arithmetic (see section 5.2).

		Petri nets	BPPs	VBPPs
linear time	linear time mu-calculus	D	D	D
	F + **EN**	U	U	U
branching time	EG	U	U	U
	EF	U	D	D
	EF + **Pres**	U	D	D

The most relevant conclusion is the existence of a ‘decidability gap’ between linear time and branching time logics. The linear time mu-calculus is decidable for an expressive model like Petri nets (when we consider only closed formulas), while EG is undecidable for a very weak model like VBPPs. The linear time mu-calculus is a powerful linear time logic, strictly more expressive than LTL, the linear time logic most commonly used for verification [28, 35]. On the contrary, EG is a weak branching time logic. Figure 2 shows a hierarchy of branching time temporal logics in order of decreasing expressive power (when considered as action-based logics). All the logics above EG are strictly more expressive.

This ‘decidability gap’ does not exist for sequential models, like context-free processes or pushdown automata. For them, the modal mu-calculus is decidable, and the modal mu-calculus is known to be strictly more expressive than the linear time mu-calculus.

After the undecidability result for EG and VBPPs, the only logic of Figure 2 which remains to be investigated is EF. We have shown that it is undecidable for arbitrary Petri nets, but decidable for BPPs and other classes of nets with semilinear reachability relations. These include persistent and reversible nets, among others (see [16]).

Finally, we have studied the addition of atomic sentences to linear time logics. If the linear time logic with F (eventually) as only operator is enriched with atomic sentences able to express which actions are enabled at a state, then it becomes undecidable for VBPPs. Loosely speaking, the addition of these atomic sentences make linear time logics as undecidable as branching time logics. Therefore, the gap between linear time and branching time time logics is in fact a gap between linear time and *one-step* branching time logics, i.e., logics which can look only one step ahead into the branching structure of the transition system.

Acknowledgements

Many thanks to Mads Dam, Colin Stirling and Astrid Kiehn for helpful discussions, and to two anonymous referees for their detailed comments.

References

[1] J.L. Balcázar, J. Díaz and J. Gabarró: Structural Complexity I, EATCS Monographs on Theoretical Computer Science 11 (1988).

[2] M. Ben-Ari, Z. Manna and A. Pnueli: The Temporal Logic of Branching Time. Acta Informatica 20(3), 207–226 (1983).

[3] G.S. Boolos and R.C. Jeffrey: Computability and Logic. Cambridge University Press (1989).

[4] J. C. Bradfield: Verifying Temporal Properties of Systems. Birkhäuser (1991).

[5] O. Burkart and B. Steffen: Model checking for context-free processes. Proceedings of CONCUR '92, LNCS 630, 123–137 (1992).

[6] O. Burkart and B. Steffen: Pushdown Processes: Parallel Composition and Model Checking. Proceedings of CONCUR '94, LNCS 836, 98-113 (1994). (1994).

[7] S. Christensen: Decidability and Decomposition in Process Algebras, Ph. D. Thesis, University of Edinburgh, CST-105-93 (1993).

[8] S. Christensen, Y. Hirshfeld and F. Moller: Bisimulation equivalence is decidable for basic parallel processes. CONCUR '93, LNCS 715, 143–157 (1993).

[9] S. Christensen, H. Hüttel and C. Stirling: Bisimulation Equivalence is decidable for all context-free processes. Proceedings of CONCUR '92, LNCS 630, 138–147 (1992).

[10] E.M. Clarke, E.A. Emerson and A.P. Sistla: Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications. ACM Transactions on Programming Languages and Systems, 8(2), pp. 244–263 (1986).

[11] M. Dam: Fixpoints of Büchi automata. Proceedings of the 12th International Conference of Foundations of Software Technology and Theoretical Computer Science, LNCS 652, 39–50 (1992).

[12] E.A. Emerson and Srinivasan: Branching Time Temporal Logic. LNCS 354, 123–172 (1988).

[13] E.A. Emerson and C. S. Jutla: Tree Automata, Mu-Calculus and Determinacy, Proceedings of FOCS '91 (1991).

[14] J. Esparza: Petri Nets, Commutative Context-Free Grammars, and Basic Parallel Processes. Proceedings of FCT '95, LNCS 965, 221-232 (1995).

[15] J. Esparza: On the decidability of model checking for several mu-calculi and Petri nets. Proceedings of CAAP '94, LNCS 787, 115–129 (1994).

[16] J. Esparza and M. Nielsen: Decidability issues for Petri nets – a survey. Bulletin of the EATCS 52, 245–262 (1994).

[17] M. J. Fischer and M.O. Rabin: Super-exponential complexity of Presburger arithmetic. Complexity of Computation, Proceedings of the SIAM-AMS Symposium on Applied Mathematics (1974).

[18] Y. Hirshfeld: Petri Nets and the Equivalence Problem. Proceedings of CSL '93, LNCS 832, 165–174 (1994).

[19] R.R. Howell and L.E. Rosier: Problems concerning fairness and temporal logic for conflict-free Petri nets. Theoretical Computer Science 64, 305–329 (1989).

[20] R.R. Howell, L.E. Rosier and H. Yen: A taxonomy of fairness and temporal logic problems for Petri nets. Theoretical Computer Science 82, 341–372 (1991).

[21] H. Hungar and B. Steffen: Local Model Checking for Context-Free Processes. Proceedings of ICALP '93, LNCS 707, (1993).

[22] H. Hungar: Model Checking of Macro Processes. Proceedings of CAV '94, LNCS 818, 169–182 (1994).

[23] P. Jančar: Decidability of a temporal logic problem for Petri nets. Theoretical Computer Science 74, 71–93 (1990).

[24] P. Jančar: Decidability Questions for Bisimilarity of Petri Nets and Some Related Problems. Proceedings of STACS '94, LNCS 775, 581–594 (1994). To appear in Theoretical Computer Science.

[25] M. Jantzen and R. Valk: The Residue of Vector Sets with Applications to Decidability Problems in Petri Nets. Acta Informatica 21, 643–674 (1985)

[26] A. Kiehn and M. Hennessy: On the Decidability of Non-Interleaving Process Equivalences. Proceedings of CONCUR '94, LNCS 836, 18–33 (1994).

[27] S.R. Kosaraju: Decidability of reachability in vector addition systems. Proceedings of the 6th Annual ACM Symposium on the Theory of Computing, 267–281 (1982).

[28] O. Lichtenstein and A. Pnueli: Checking that finite state programs satisfy their linear specifications. Proceedings of the 12th ACM Symposium on Principles of Programming Languages, 97–107 (1985).

[29] R. Lipton: The Reachability Problem Requires Exponential Space. Technical Report 62, Yale University (1976).

[30] E.W. Mayr: An algorithm for the general Petri net reachability problem. SIAM Journal of Computing 13, 441-460 (1984).

[31] M. Minsky: Computation: Finite and Infinite Machines. Prentice-Hall (1967).

[32] D. Muller and P. Schupp: The Theory of Ends, Pushdown Automata and Second Order Logic. Theoretical Computer Science 37, 51–75 (1985).

[33] J.L. Peterson: Petri Net Theory and the Modelling of Systems. Prentice-Hall (1981).

[34] C. P. Stirling: Modal and temporal logics. In Handbook of Logic in Computer Science, Vol. 2, Oxford University Press, 477–563 (1992).

[35] M.Y. Vardi and P. Wolper: Automata Theoretic Techniques for Modal Logics of Programs. Journal of Computer and System Sciences 32, 183–221 (1986).

[36] H. Yen: A Unified Approach for Deciding the Existence of Certain Petri Net Paths. Information and Computation 96(1), 119–137 (1992).