Decidability of model checking for infinite-state concurrent systems

Javier Esparza
Laboratory for Foundations of Computer Science
University of Edinburgh

August 26, 1994

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 are only applicable to the finite state case, in which the semantics of the system is a finite object, like a finite transition system. However, many interesting systems have infinite state spaces, and these techniques are not applicable.

Several verification problems have been recently shown to be decidable for some classes of processes with infinite state spaces. These results can be classified into those showing the decidability of equivalence relations [8, 7, 26], and those showing the decidability of the 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 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) [29, 12]. Simpler algorithms have been given for the alternation-free fragment of the mu-calculus [4, 22]. These results have been extended to context-free like processes in [23], and to pushdown processes in [5].

The model checking problem for Petri nets was first studied by Howell and Rosier [20], who observed that a simple 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 $ge(s, c)$ (with intended meaning 'at the current marking, the number of tokens on place s is greater than or equal to c') or of type $fi(t)$ (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 occurrence sequence that satisfies it.

In a subsequent paper [21], 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 [24] that the positive fragment with **GF** (eventually always) as operator, instead of **F**, is decidable as well. More recently, Bradfield showed that the model checking problem for (closed formulae of) the modal mu-calculus is undecidable [3].

Although some of these results are very deep, they are also rather fragmentary. The logics to be studied 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 [21, 24] is to study fairness problems), and not the logics themselves. Branching temporal logics seem not to have been considered yet, and the only result about mu-calculi is the one by Bradfield. In this paper, we intend to carry out a more comprehensive study.

Another problem is that the logics of [20, 21, 24] mix state-based and event-based atomic sentences (state-based atomic sentences express properties of the current marking, while event-based sentences express properties of the actions that can occur next), while the logics for sequential models of [4, 22] only have event-based sentences. This makes it difficult to compare results for Petri nets and for sequential models. In this paper, we study event-based logics, and in the last section comment on how the addition of state-based atomic sentences affects (sometimes dramatically) the results.

We shall consider interleaving semantics and, essentially, two models of computation: labelled Petri nets, called just Petri nets in the sequel, and Basic Parallel Processes (BPPs) [6]. 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 has been proposed so far (context sensitive grammars lie strictly between Petri nets and Turing Machines [30], 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, which extends finite automata with a parallel operator. 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 hold for any other infinite state concurrent model. Our main undecidability result will hold even for a subclass of BPPs, which strenghtens this argument.

The paper is organised as follows. In the first section, we define Petri nets and BPPs. Section 2 deals with linear time logics, and Section 3 with branching time logics. Section 4 shows how the results of sections 2 and 3 change when state-based atomic sentences are added.

2 Models: notations and basic definitions

2.1 Petri Nets

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

- S and T are two disjoint, finite sets,

- $W\colon (S \times T) \cup (T \times S) \to I\!N$ 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*.

Given a node x of N, ${}^\bullet x = \{y \mid (y,x) \in F\}$ is the *preset* of x and $x^\bullet = \{y \mid (x,y) \in F\}$ is the *postset* of x.

Given a set of nodes X of N, we define ${}^\bullet X = \bigcup_{x \in X} {}^\bullet x$ and $X^\bullet = \bigcup_{x \in X} x^\bullet$.

A *marking* of N is a mapping $M\colon S \to I\!N$. 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 expression $M_1 \xrightarrow{a} M_2$, where M_1, M_2 are markings of N, denotes that M_1 enables some transition t labelled by a, and that the marking reached by the occurrence of t is M_2. A sequence $M_0 \xrightarrow{a_1} M_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} M_n$ is a *finite occurrence sequence* leading from M to M_n and we write $M_0 \xrightarrow{a_1 \ldots a_n} M_n$. We say that M_n is *reachable* from M_0, or just reachable in Σ. The empty sequence ϵ is an occurrence sequence: we have $M \xrightarrow{\epsilon} M$ for every marking M. A sequence $M_0 \xrightarrow{a_1} M_1 \xrightarrow{a_2} \cdots$ is an *infinite occurrence sequence*. We write $M \xrightarrow{a_1\, a_2 \ldots}$.

We assume that if two transitions have the same label, then they do not have the same preset and postset. Under this assumption, every label of an occurrence sequence can be uniquely mapped to its underlying transition. Making implicit use of this property, we sometimes speak about the transitions that occur in an occurrence sequence.

A sequence of actions σ is *enabled* at a marking M if $M \xrightarrow{\sigma} M'$ for some marking M' (if σ is finite) or $M \xrightarrow{\sigma}$ (if σ is infinite).

An occurrence sequence is *maximal* if either it is infinite or it leads to a dead marking. The *language* of Σ, denoted by $L(\Sigma)$, is the set of words obtained by dropping the intermediate markings in the maximal occurrence sequences of Σ.

The *reachability graph* of a Petri net Σ is the labelled graph having the reachable markings of Σ as nodes; there is an edge between two nodes M and M' labelled by a if $M \xrightarrow{a} M'$.

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 language and reachability graph of a Petri net carries over to unlabelled Petri nets.

2.2 Basic and Very Basic Parallel Processes

The class of Basic Parallel Process (BPP) expressions is defined by the following abstract syntax:

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

where a belongs either to a set of *atomic actions* or is the *silent action* τ. The BPP expressions containing no occurrence of the choice operator $+$ are called Very Basic Parallel Process (VBPP) expressions.

A BPP is defined by 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 E_i is *guarded*, that is, appear 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 the *leading equation*.

Any BPP determines a labelled transition system, whose transition relations $\stackrel{a}{\longrightarrow}$ are the least relations satisfying the following rules:

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

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

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

A *computation* of a BPP is either an infinite sequence $E_1 \stackrel{a_1}{\longrightarrow} E_2 \stackrel{a_2}{\longrightarrow} E_3 \ldots$ or a finite sequence $E_1 \stackrel{a_1}{\longrightarrow} \ldots \stackrel{a_n}{\longrightarrow} E_n$ such that $E_n \stackrel{a}{\not\longrightarrow}$ for any action a. We say that E_n is *reachable* from X_1. The computations of the BPPs constructed in this paper will always be infinite.

The *labelled transition system* associated to a BPP with leading variable X_1 is the labelled graph having the processes E reachable from X_1 as nodes, and $\bigcup_{a \in Act} \stackrel{a}{\longrightarrow}$ as edges.

A BPP is in *standard form* if every expression E_i on the right hand side of an equation is of the form $a_1.\alpha_1 + \ldots + a_n.\alpha_n$, where for each i the expression α_i is a merge of process variables. It is shown in [6] that every BBP is strongly bisimilar to a BPP in standard form.

Every BPP in standard 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 E_i. For each subexpression $a_j \alpha_j$ in the defining equation of E_i, a transition is added having the place E_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 the weight n. It is easy to see that the transition system of a BPP in standard form and the reachability graph of its associated Petri net are isomorphic.

Also, it follows easily from this translation that:

- the transitions of Petri nets corresponding to BPPs in standard form have exactly one input place, and

Fig. 1 A BPP and its corresponding Petri net

- the places of VBPPs in standard form have at most one output transition.

Moreover, all the arcs leading from places to transitions have weight 1.

3 Linear Time Logics

3.1 The linear time μ-calculus

The linear time μ-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 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:

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

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 $\|\phi\|$. We also use the following abbreviations: $\mathbf{true} = \nu Z.Z$, $\phi_1 \vee \phi_2 = \neg(\neg\phi_1 \wedge \neg\phi_2)$, and $\mu Z.\phi = \neg\nu Z.\phi[\neg Z/Z]$. Observe that $\|\mathbf{true}\| = Act^\infty$.

For a more detailed description of the linear-time mu-calculus the reader is referred to [10].

Let Σ be a labelled Petri net and ϕ a closed formula of the linear time μ-calculus. We say that Σ satisfies ϕ if $L(\Sigma) \subseteq \|\phi\|$. 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 [10].

3.2 Decidability of the linear time μ-calculus for Petri nets

We define the model checking problem for the linear time μ-calculus as follows: given a Petri net Σ and a closed formula ϕ, determine if Σ satisfies ϕ or not. We prove in this section that this problem is decidable. The decision procedure is based on an automata-theoretic characterisation of the logic [32]. We show that there exist two automata $A_{\neg\phi}$ and $B_{\neg\phi}$ that accept the finite and infinite words of $\|\neg\phi\|$, respectively. It follows that Σ satisfies ϕ if and only if $L(\Sigma) \cap L(A_{\neg\phi}) = \emptyset$ and $L(\Sigma) \cap L(B_{\neg\phi}) = \emptyset$. To decide these two properties, we construct Petri net representations of $A_{\neg\phi}$ and $B_{\neg\phi}$, and combine them with Σ in a way similar to the product of automata. It is then easy to show that the two properties are equivalent to two decidable properties of Petri nets.

An automaton over an alphabet Act is a fourtuple $(Q, q_0, \delta, \mathcal{F})$, where Q is the set of states, q_0 the initial state, $\delta: Q \times Act \to Q$ the transition function and $\mathcal{F}$ the set of final states. Finite and Büchi automata are automata with different acceptance conditions: a finite automaton A accepts a word w in $Act^\star$ if the computation of A on w ends in some final state; a Büchi automaton B accepts a word w in Act^ω if the computation of B on w passes infinitely often through some final state.

Dam provides in [10] a procedure to construct, given a formula ϕ of a different version of the linear time μ-calculus, a Büchi automaton whose language is the denotation of the formula. Dam's version has one single next operator O, instead of an operator (a) for every action a. Given a finite set V of propositional variables, the denotation of a formula with free variables in V is a set of words over the alphabet 2^V. The rules defining the denotation of a formula are like the ones given above for $\neg$, $\wedge$ and greatest fixpoints, plus the following rule for the new next operator:

$$\|O\phi\|_{\mathcal{V}} \quad = \quad \{\sigma \in Act^\infty \mid \sigma^1 \in \|\phi\|_{\mathcal{V}}\}$$

We briefly discuss how to adapt Dam's construction to our case. Dam's automaton has $\mathcal{P}(V) \times \mathcal{P}(V)$ as alphabet, where $\mathcal{P}(V)$ is the powerset of V. The automaton is constructed in a compositional way. A word w is accepted by the automaton if and only if there is an accepting run

$$q_0 \xrightarrow{(\alpha_1^+, \alpha_1^-)} q_1 \xrightarrow{(\alpha_2^+, \alpha_2^-)} \cdots$$

such that at the ith point of w all variables in α_i^+ hold and no variable in α_i^- holds.

For our version of the linear time μ-calculus, we enrich the alphabet of the automaton to $\mathcal{P}(V) \times \mathcal{P}(V) \times (Act \cup \{\tau\})$, where $\tau \notin Act$. Dam's construction can now be easily modified to take this third component of the alphabet into account. A word w is accepted by the new automaton if and only if there is an accepting run

$$q_0 \xrightarrow{(\alpha_1^+,\alpha_1^-,a_1)} q_1 \xrightarrow{(\alpha_2^+,\alpha_2^-,a_2)} \cdots$$

such that at the ith point of w all variables in α_i^+ hold, no variable in α_i^- holds, and the third component of w at this point is a_i.

If ϕ is a closed formula, we choose $V = \emptyset$; then, the automaton we obtain has just $Act \cup \{\tau\}$ as alphabet. The rôle of the action τ is the following: in our case, the Büchi automaton does not accept exactly $||\phi||$, because $||\phi||$ may contain finite words (for instance, $a \in ||(a)\,\mathbf{true}||$), while a Büchi automaton only accepts infinite ones. If $||\phi||$ contains a finite word w, then the automaton accepts $w\,\tau^\omega$. The action τ only appears in transitions of the form (q, τ, q).

Proposition 3.1

Let ϕ be a closed formula of the linear time μ-calculus over a a set of actions Act. There exist a Büchi automaton B with alphabet $Act \cup \{\tau\}$ which accepts a word w iff $w \in ||\phi|| \cap Act^\omega$ or $w = w'\tau^\omega$ and $w' \in ||\phi|| \cap Act^\star$.

Proof: Use Dam's construction with the modifications discussed above. ■

It is convenient for our analysis to split the automaton B of Proposition 3.1 into a finite automaton which accepts the finite words of $||\phi||$ and a Büchi automaton which accepts the infinite ones.

Proposition 3.2

Let ϕ be a closed formula of the linear time μ-calculus over a set of actions Act. There exist a finite automaton A_ϕ and a Büchi automaton B_ϕ, both with alphabet Act, such that $L(A_\phi) = ||\phi|| \cap Act^\star$, and $L(B_\phi) = ||\phi|| \cap Act^\omega$.

Proof: A_ϕ and B_ϕ are the same automaton (i.e., they differ only in the acceptance condition), obtained by just removing the transitions of the form (q, τ, q) from δ. ■

We now assign to a given automaton a Petri net. Let $A = (Q, q_0, \delta, \mathcal{F})$ be an automaton over an alphabet Act. The labeled Petri net $\Sigma^A = (S, T, W, M_0, l)$ – with labels on Act – is defined as follows:

$$\begin{aligned}
S &= Q \\
T &= \delta \\
W(q, (q', a, q'')) &= \begin{cases} 1 & \text{if } q = q' \\ 0 & \text{otherwise} \end{cases} \\
W((q', a, q''), q) &= \begin{cases} 1 & \text{if } q'' = q \\ 0 & \text{otherwise} \end{cases} \\
M_0(q) &= \begin{cases} 1 & \text{if } q = q_0 \\ 0 & \text{otherwise} \end{cases} \\
l((q, a, q')) &= a
\end{aligned}$$

It follows immediately from this construction that the finite automaton A_ϕ accepts a word σ if and only if σ is an occurrence sequence of the Petri net Σ^{A_ϕ} leading to a marking in which some final state is marked (recall that the places of Σ^{A_ϕ} are the states of A). Similarly, B_ϕ accepts a word σ if and only if σ is an infinite occurrence sequence such that infinitely many intermediate markings along the sequence put one token in some of the final states of B_ϕ. If $T_{\mathcal{F}}$ is the set of input transitions of these states, then an equivalent condition is that σ contains infinitely many occurrences of some transition of $T_{\mathcal{F}}$.

Let $\Sigma_1 = (S_1, T_1, W_1, l_1, M_{01})$, $\Sigma_2 = (S_2, T_2, W_2, l_2, M_{02})$ be two labelled Petri nets with disjoint sets of nodes. The labelled Petri net $\Sigma_1 \times \Sigma_2 = (S, T, W, l, M_0)$ is defined as follows:

$$
\begin{aligned}
S &= S_1 \cup S_2 \\
T &= \{\{t_1, t_2\} \mid t_1 \in T_1 \wedge t_2 \in T_2 \wedge l_1(t_1) = l_2(t_2)\} \\
W(s, \{t_1, t_2\}) &= \begin{cases} W_1(s, t_1) & \text{if } s \in S_1 \\ W_2(s, t_2) & \text{if } s \in S_2 \end{cases} \\
W(\{t_1, t_2\}, s) &= \begin{cases} W_1(t_1, s) & \text{if } s \in S_1 \\ W_2(t_2, s) & \text{if } s \in S_2 \end{cases} \\
M_0(s) &= \begin{cases} M_{01}(s) & \text{if } s \in S_1 \\ M_{02}(s) & \text{if } s \in S_2 \end{cases} \\
l(\{t_1, t_2\}) &= l_1(t_1)
\end{aligned}
$$

Observe that this product operation is very similar to the usual product of automata.

Every marking M of $\Sigma_1 \times \Sigma_2$ projects onto a marking M_1 of Σ_1 and a marking M_2 of Σ_2; moreover, these two projections determine M. Making use of this property, we denote $M = (M_1, M_2)$. So, in particular, $M_0 = (M_{01}, M_{02})$.

The following lemma is an immediate consequence of the definitions and the occurrence rule for Petri nets.

Lemma 3.3

$(M_1, M_2) \xrightarrow{\sigma} (M_1', M_2')$ iff $M_1 \xrightarrow{\sigma} M_1'$ and $M_2 \xrightarrow{\sigma} M_2'$ (where M_1, M_1' are markings of Σ_1 and M_2, M_2' markings of σ_2).

Proof: ($\Rightarrow$): Let $\{t_1, u_1\}\{t_2, u_2\}\ldots$ be the sequence of transitions of $\Sigma_1 \times \Sigma_2$ underlying σ. Then, $t_1\, t_2 \ldots$ and $u_1\, u_2 \ldots$ are sequences of transitions underlying the occurrence sequence σ in Σ_1 and Σ_2, respectively.
($\Leftarrow$): Similar to ($\Rightarrow$). ∎

We can now reduce $L(\Sigma) \cap L(B_\phi) \neq \emptyset$ to a property of the Petri net $\Sigma \times \Sigma^{B_\phi}$.

Lemma 3.4

$L(\Sigma) \cap L(B_\phi) \neq \emptyset$ iff $\Sigma \times \Sigma^{B_\phi}$ has an occurrence sequence which contains infinitely many occurrences of some transition $\{t, u\}$, where $u \in T_{\mathcal{F}}$.

Proof: ($\Rightarrow$): Let σ be a word of $L(\Sigma) \cap L(B_\phi)$. Then σ is the sequence of labels of two occurrence sequences in Σ and Σ^{B_ϕ}; moreover, σ contains infinitely many occurrences of some transition u of $T_{\mathcal{F}}$. By Lemma 3.3, σ is the sequence of labels of an occurrence sequence in $\Sigma \times \Sigma^{B_\phi}$, in which transitions of the form $\{t, u\}$ occur infinitely often.
($\Leftarrow$): Let $(M_{01}, M_{02}) \xrightarrow{\sigma}$ be an infinite occurrence sequence of $\Sigma \times \Sigma^{B_\phi}$ which contains infinitely many occurrences of of some transition $\{t, u\}$, where $u \in T_{\mathcal{F}}$. By Lemma 3.3, $M_{02} \xrightarrow{\sigma}$ is an occurrence sequence of Σ^{B_ϕ}. Let $M_{02} \xrightarrow{a_1} M_1 \xrightarrow{a_2} M_2 \xrightarrow{a_3} \ldots$ be the

full representation of $M_{02} \xrightarrow{\sigma}$. By the definition of Σ^{B_ϕ}, each marking M_i puts one token in exactly one place of Σ^{B_ϕ}, and no tokens in the rest. So each marking M_i is univoquely associated to a state of B_ϕ, and the whole sequence to a computation. The state associated to the markings succeeding an occurrence of $\{t, u\}$ is a final state of B_ϕ; so B_ϕ accepts σ. ∎

The condition on $\Sigma \times \Sigma^{B_\phi}$ of Lemma 3.4 was shown to be decidable by Jantzen and Valk [25]. Yen shows in [33] that it is decidable within exponential space.

Lemma 3.5 *[33]*

Let Σ be a Petri net with n nodes, and let T_0 be a subset of transitions of Σ. It can be decided in $O(2^{c \cdot n \cdot \log n})$ space, for some constant c, if some infinite occurrence sequence of Σ contains infinitely many occurrences of some transition of T_0. ∎

Proof: There exists such an infinite occurrence sequence iff there exists a finite occurrence sequence $M_0 \xrightarrow{\sigma_1} M_1 \xrightarrow{\sigma_2} M_2$ such that $M_1 \leq M_2$ and σ_2 contains some occurrence of a transition of T_0 (then M_1 enables the sequence σ_2^ω). This can be expressed as a formula on paths of Petri nets which belongs to the class of formulas defined by Yen in [33]. Yen shows that any of these formulas can be decided using the space indicated in the lemma. ∎

$L(\Sigma) \cap L(A_\phi) \neq \emptyset$ can also be reduced to a suitable property of $\Sigma \times \Sigma^{A_\phi}$.

Lemma 3.6

$L(\Sigma) \cap L(A_\phi) \neq \emptyset$ iff there exists a reachable dead marking of $\Sigma \times \Sigma^{A_\phi}$ which puts one token in some final state of A_ϕ.

Proof: ($\Rightarrow$): Let σ be a word of $L(\Sigma) \cap L(A_\phi)$. Then, $M_{01} \xrightarrow{\sigma} M_1$ in Σ and $M_{02} \xrightarrow{\sigma} M_2$ in Σ^{A_ϕ}; moreover, M_1 is a dead marking of Σ, and M_2 puts a token in some final state of A_ϕ. By Lemma 3.3, $(M_{01}, M_{02}) \xrightarrow{\sigma} (M_1, M_2)$ is an occurrence sequence of $\Sigma \times \Sigma^{A_\phi}$. Since M_1 is a dead marking of Σ, (M_1, M_2) is a dead marking of $\Sigma \times \Sigma^{A_\phi}$.
($\Leftarrow$): Let $(M_{01}, M_{02}) \xrightarrow{\sigma} (M_1, M_2)$ be an occurrence sequence of Σ such that (M_1, M_2) is a dead marking, and M_2 puts a token in some final state of A_ϕ. By Lemma 3.3, $M_{02} \xrightarrow{\sigma} M_2$ is an occurrence sequence of Σ^{A_ϕ}, and therefore $\sigma \in L(A_\phi)$.

It remains to prove that σ belongs to the language of Σ, i.e., that $M_{01} \xrightarrow{\sigma} M_1$ is a maximal occurrence sequence of Σ.

Assume this is not the case. Then there exists an occurrence sequence $M_{01} \xrightarrow{\sigma} M_1 \xrightarrow{a} M_1'$. Let t be the transition underlying the occurrence of a. By the definition of A_ϕ, a belongs to the alphabet of A_ϕ, and therefore to the alphabet of Σ^{A_ϕ} as well. So Σ^{A_ϕ} contains a transition (q, a, q'), where q is the final state marked at M_2. This transition is enabled at M_2. By Lemma 3.3, $\{t, (q, a, q')\}$ is enabled at (M_1, M_2), which contradicts that (M_1, M_2) is a dead marking. ∎

The existence of the dead marking of Lemma 3.6 can be decided by solving an exponential number of instances of the *submarking reachability problem* [18]. Given a net with a set of places S, a *submarking* of the net is a *partial* mapping from S onto $\mathbb{N}$. The submarking reachability problem is the problem of deciding, given a Petri net $\Sigma = (N, M_0)$ and a submarking P of N, if some reachable marking coincides with P on all the places where P is defined. The submarking reachability problem is reducible (in polynomial time) to the reachability problem [18].

Lemma 3.7

Let Σ be a Petri net, and let S_0 be a subset of places of Σ. It is decidable if some dead marking of Σ puts a token on some place of S_0.

Proof: If M is a dead marking, then for every transition t some input place of t is unmarked at M. The set of submarkings which specify that, for every transition t, some input place of t is unmarked, and moreover that some place of S contains one token, is finite: it contains at most $|S|^{|T|} \cdot |S_0|$ elements, where S, T are the sets of places and transitions of the net. The property of the lemma can then be decided by solving the submarking reachability problem for these sets. ■

The complexity of the reachability problem is still open. The most efficient algorithm is not primitive recursive, while the best known lower bound is exponential space [27]. If exponential space suffices, then the complexities of deciding $L(\Sigma) \cap L(B_\phi) = \emptyset$ and $L(\Sigma) \cap L(A_\phi) = \emptyset$ are similar. Otherwise, deciding the second property is more involved.

Theorem 3.8

Let Σ be a labelled Petri net, and let ϕ be a closed formula of the linear time μ-calculus. It is decidable if Σ satisfies ϕ.

Proof: Σ satisfies ϕ iff $L(\Sigma) \subseteq ||\phi||$, or, by the semantics of negation, iff $L(\Sigma) \cap ||\neg\phi|| = \emptyset$.
By Proposition 3.2, $L(\Sigma) \cap ||\neg\phi|| = \emptyset$ iff $L(\Sigma) \cap L(A_{\neg\phi}) = \emptyset$ and $L(\Sigma) \cap L(B_{\neg\phi}) = \emptyset$. Use then Lemmas 3.4, 3.5, 3.6 and 3.7. ■

4 Branching Time Logics

The decidability border changes rather drastically when we move from linear time to branching time logics. We have seen that powerful linear-time logics like the linear-time mu-calculus are decidable for powerful models like Petri nets. In this section we shall see that rather weak logics, like the Unified System of Branching Time (UB) [1] are 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.

4.1 The modal mu-calculus

The modal μ-calculus has the same syntax as the linear time μ-calculus, just substituting $\langle a \rangle\phi$ for $(a)\phi$, but 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:

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

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 abbreviations **true** and $\mu Z.\phi$, defined as for the linear time mu-calculus, and $[a]\phi =\neq \langle a\rangle\neg\phi$.

As in the case of the linear time μ-calculus, the denotation of a closed formula is independent of the valuation.

To define when a Petri net $\Sigma = (N, M_0)$ satisfies a closed formula ϕ, we consider its reachability graph as a rooted, labelled transition system. The root is the marking M_0. We then say that Σ satisfies ϕ if $M_0 \in ||\phi||_{\mathcal{V}}$.

For details about the modal mu-calculus and its expressive power, see [31].

4.2 Undecidability of the modal mu-calculus for VBPPs

We prove the undecidability of the model checking problem by means of a reduction to the halting problem of counter machines [28] (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 $(0 \leq i \leq n)$. The states are divided into two types. The states of type I have transition rules of the form transition rule

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

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 [28].

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

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

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 computations, some of which – the 'honest' computations – simulate the computation of the counter machine, while the rest are 'dishonest' computations 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 computation 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 $\mathtt{C_j}$.

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

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

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

A state q_i of type II is modelled by the processes

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

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

$$\begin{aligned} \mathtt{SQ_{n+1}} &\stackrel{\text{def}}{=} \mathtt{in_{n+1}} \cdot (\mathtt{SQ_{n+1}} \parallel \mathtt{Q_{n+1}}) \\ \mathtt{Q_{n+1}} &\stackrel{\text{def}}{=} \mathtt{halt} \cdot \mathbf{0} \end{aligned}$$

Finally, the model `M` of the counter machine is defined as follows

$$\begin{aligned} \mathtt{SM} &\stackrel{\text{def}}{=} (\mathtt{SQ_1} \parallel \ldots \parallel \mathtt{SQ_{n+1}}) \parallel \mathtt{SC_1} \parallel \ldots \parallel \mathtt{SC_m}) \\ \mathtt{M} &\stackrel{\text{def}}{=} \mathtt{SM} \parallel \mathtt{Q_0} \end{aligned}$$

It follows easily from the operational semantics of BPPs that the reachable states of `M` have 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 $\mathtt{P}^k$ is defined as $\underbrace{(P \parallel \ldots \parallel P)}_{k}$. The states in which all the indices $i_0, \ldots, i_{n+1}$ but one are 0 correspond to the configurations of the counter machine. The nonzero index indicate the state, and the indices $j_1, \ldots, j_m$ the values of the counters. We say that these states are *meaningful*.

We can now define the *honest* computations of $\mathtt{M}$ as those containing a prefix with the following property: the projection of the sequence of states reached along the prefix on the set of meaningful states corresponds to the computation of the counter machine $\mathcal{M}$.

It is clear that $\mathtt{M}$ has honest computations, but not every computation of $\mathtt{M}$ is honest.

We now construct a formula of the mu-calculus expressing that a certain honest computation eventually executes $\mathtt{halt}$. For a state q_i of type I we define the open formula

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

For a state q_i of type II we define

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

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

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

Loosely speaking, a computation of $\mathtt{M}$ satisfies this formula if and only if it eventually executes the action $\mathtt{halt}$, and the prefix preceeding the first occurrence of $\mathtt{halt}$ is the concatenation of small sequences of actions, each of which satisfies one of the formulae ϕ_i. Each of these small sequences simulates one step of the counter machine.

Theorem 4.1

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

Proof: Let $\mathcal{M}$ be a counter machine with initially empty counters. The VBPP $\mathtt{M}$ satisfies the formula *Halt* iff it has a honest computation that eventually executes the action $\mathtt{halt}$. Since honest computations 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 very small subset of the modal mu-calculus, namely the Unified System of Branching Time [1].

4.3 Unified System of Branching Time

The Unified System of Branching Time (UB) is a logic introduced by Ben-Ari, Manna and Pnueli [1], 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 event 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 by the execution of a. Also, we only allow the atomic proposition **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 [9], which in turn can be easily embedded in the modal mu-calculus (see, for instance, [3]).

We interpret UB formulae on a labelled transition system $\mathcal{T} = (\mathcal{S}, \bigcup_{a\in\Lambda} \xrightarrow{a})$. A *path* is a maximal sequence $(s_0, s_1, \ldots)$ of states such that for every $i \geq 0$ there exists some action a such that $s_i \xrightarrow{a} s_{i+1}$.

Loosely speaking, the formulae $EF\phi$ and $EG\phi$ hold at a state s_0 if there exists a path starting at s_0 such that the property ϕ holds at some - respectively, at every - state.

We denote that a state s_0 satisfies a formula ϕ by $s_0 \models \phi$. The satisfaction relation is inductively defined as follows:

$$\begin{array}{lll}
s_0 \models \mathbf{true} & & \text{always} \\
s_0 \models \neg\phi & \text{iff} & \text{not}(s_0 \models \phi) \\
s_0 \models \phi_1 \wedge \phi_2 & \text{iff} & s_0 \models \phi_1 \text{ and } s_0 \models \phi_2 \\
s_0 \models E(a)\phi & \text{iff} & \text{for some } s \text{ such that } s_0 \xrightarrow{a} s,\ s \models \phi \\
s_0 \models EF\phi & \text{iff} & \text{for some path } (s_0, s_1, \ldots), \exists i \geq 0\ s_i \models \phi \\
s_0 \models EG\phi & \text{iff} & \text{for some path } (s_0, s_1, \ldots), \forall j \geq 0\ s_j \models \phi))
\end{array}$$

We define the following dual operators:

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

In particular, $AF\phi$ holds at a state if every path starting at it contains a state satisfying ϕ.

A BPP with leading variable X satisfies a formula ϕ if $X \models \phi$. The model checking problem consists in deciding if $X \models \phi$.

4.4 Undecidability of UB for VBPPs

In order to prove the undecidability of the model checking problem, we shall find a formula ϕ_h such that:

(1) there exist some computation such that all the states reached along its execution satisfy ϕ_h, and

(2) if all the states reached along a computation satisfy ϕ_h, then the computation is honest.

Let us see that, once this formula has been found, the undecidability result follows easily. We define the formula

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

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

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

Unfortunately, it is not possible to find the formula ϕ_h for the model $\mathtt{M}$ of a counter machine we described in the case of the mu-calculus. The essence of the problem is easier to see in a simpler process $\mathtt{P}$ given by

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

Assume that $\mathtt{P}$ is simulating, in a weak sense, a certain system. Assume further that the honest computations of $\mathtt{P}$ are those whose prefixes satisfy $|\#\mathtt{q} - \#\mathtt{r}| \leq 1$, where $\#\mathtt{q}$, $\#\mathtt{r}$ is the number of $\mathtt{q}$'s and $\mathtt{r}$'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 computation $E_0 \xrightarrow{\mathtt{a}_1} E_1 \xrightarrow{\mathtt{a}_2} \ldots$ of $\mathtt{P}$, we have $\mathtt{P} = E_0 = E_1 = \ldots$. Therefore, all intermediate states satisfy the same properties, and no formula of UB can distinguish between honest and dishonest computations.

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

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

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

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

The solution. If we are allowed to split the actions $\mathtt{q}$ and $\mathtt{r}$ in the definition of the process $\mathtt{P}$ above, things change. Define now

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

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

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

From $\mathtt{P}$, 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 into two is not enough.

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

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

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

A state q_i of type II is modelled by

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

The other equations do not change.

To define the formula ϕ_h, we first introduce the following notation. Let A be the set of actions of the form $\mathtt{out}^1_\mathtt{i}$, $\mathtt{out}^2_\mathtt{i}$, $\mathtt{dec}^2_\mathtt{i}$ or $\mathtt{dec}^3_\mathtt{i}$, and let $a_1, \ldots, a_k$ be actions of A. Then,

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

In other 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 formulae. For each state q_i of type I, ϕ_h contains a disjunct of the form

$$\widehat{EN}(\mathtt{out_i})$$

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

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

and the second is

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

(notice the similarity with the formula that was used in our toy example).

Lemma 4.2

If all the states of a computation of the model $\mathtt{M}$ *satisfy the formula* ϕ_h*, then the computation is honest.*

Proof: Let

$$E = (\mathtt{SM} \parallel \mathtt{Q_i} \parallel \mathtt{C_1}^{i_1} \parallel \ldots \parallel \mathtt{C_n}^{i_n})$$

be a meaningful state of a computation in which every state satisfies ϕ_h. We prove that the next meaningful state of the computation is the one that models correctly one step of the counter machine, i.e., the one that corresponds to the next configuration in the computation of the counter machine. The result then follows by induction.

The proof is carried out by examining the actions enabled at E, and checking 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. The procedure terminates 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, or

(2) q_i is of type II, and the process C_j does not appear in E, or

(3) q_i is of type II, and the process C_j does appear in E.

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 $i_j = 0$.

In this case, the actions enabled at E are $\mathtt{out}_\mathtt{i}^1$, all the $\mathtt{in}$ actions, and the $\mathtt{dec}^1$ actions of the counters which are nonempty at E. The $\mathtt{in}$ actions lead to a state where either $\mathtt{out}_\mathtt{i}^1$ and some other $\mathtt{out}$ action are enabled, or the sequence $\mathtt{out}_\mathtt{i}^1\ \mathtt{out}_\mathtt{i}^1$ is enabled. Such a state does not satisfy ϕ_h. The $\mathtt{dec}^1$ actions lead to states where some $\mathtt{dec}^2$ action is enabled. They do not satisfy ϕ_h either. So the next action in the computation can only be $\mathtt{out}_\mathtt{i}^1$. Then we have $E \xrightarrow{\mathtt{out}_\mathtt{i}^1} E'$, where

$$E' = (\mathtt{SM} \parallel \mathtt{out}_\mathtt{i}^2 \cdot \mathbf{0} \parallel \mathtt{C_1}^{i_1} \parallel \ldots \parallel \mathtt{C_n}^{i_n})$$

At the new state E', the enabled actions are $\mathtt{out}_\mathtt{i}^2$, all the $\mathtt{in}$ actions, and the $\mathtt{dec}^1$ actions of the nonempty counters. The action $\mathtt{out}_\mathtt{i}^2$ leads to a state where no $\mathtt{out}$ action is enabled, and the $\mathtt{dec}^1$ actions are not possible by the same argument as above. The only possible action is $\mathtt{in_k}$, which leads to the state

$$E'' = (\mathtt{SM} \parallel \mathtt{out}_\mathtt{i}^2 \cdot \mathbf{0} \parallel \mathtt{Q_k} \parallel \mathtt{C_1}^{i_1} \parallel \ldots \parallel \mathtt{C_n}^{i_n})$$

The actions enabled at E'' are $\mathtt{out}_\mathtt{k}^1$, $\mathtt{out}_\mathtt{i}^2$ and the $\mathtt{dec}^1$ actions of the nonempty counters. It is easy to see that the only next possible action is $\mathtt{out}_\mathtt{i}^2$, which leads to the state

$$E''' = (\mathtt{SM} \parallel \mathtt{Q_k} \parallel \mathtt{C_1}^{i_1} \parallel \ldots \parallel \mathtt{C_n}^{i_n})$$

E''' is a meaningful state. Therefore, the next actions of a computation starting from the state E which satisfies ϕ_h at every state must be

$$\mathtt{out}_\mathtt{i}^1\ \ \mathtt{in_k}\ \ \mathtt{out}_\mathtt{i}^2$$

This sequence faithfully simulates the transition from q_i to q_k while keeping the counter c_j to 0.

In case (1), the only possible next action is $\mathtt{out}_\mathtt{i}$, and in case (3) the only possible sequence is

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

Again, these sequences faithfully simulate the computation of the counter machine. Therefore, the (in fact, unique) computation that satisfies ϕ_h at every state is honest. ■

Theorem 4.3

The model checking problem for the logic UB 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 $\mathtt{M}$ satisfies the formula *Halt*. ■

4.5 The logic UB$^-$

UB$^-$ is the fragment of UB in which the operator EG or its dual AF are forbidden (we follow the notation of [11]). So UB$^-$ 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' ($EG\phi$).

We show in this section that the model checking problem for the logic UB$^-$ and arbitrary Petri nets is undecidable, but that it is decidable for several classes of Petri nets, including those derived from BPPs. We also also show that in this latter case the model checking problem is PSPACE-hard.

4.6 Undecidability of UB$^-$ for arbitrary Petri nets.

We prove undecidability by a reduction from the *containment problem.* An instance of the problem is a triple $(\Sigma 1, \Sigma_2, f)$, 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. The proof can be found in [18].

Let (Σ_1, Σ_2, f) be an instance of the containment problem, and let S_1, S_2 be the sets of places 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 in several steps. At each step we add some places and transitions. We represent a transition having X and Y as sets of input and output places, respectively, as $X \rightarrow Y$. All the arcs of Σ have weight 1, and therefore with this notation we specify at the same time which is the set of transitions and which is the weighting function. To give a transition $X \rightarrow Y$ the name t, we write $t: X \rightarrow Y$. The steps are:

- Add $\Sigma_1 \cup \Sigma_2$.
- Add three places A, B, C, and two transitions

$$\begin{aligned} t_{AB} &: \{A\} \cup S_1 \rightarrow \{B\} \cup S_1 \\ t_{BC} &: \{B\} \cup S_2 \rightarrow \{C\} \cup S_2 \end{aligned}$$

Put one token on A, and no tokens on B and C.

- For every place s of Σ_1, add a transition $\{s, f(s), C\} \to \{C\}$.
- For every place s added so far, with the exception of C, add the transition $\{s\} \to \{s\}$.

Every marking M of Σ projects onto a marking M_1 of Σ_1, a marking M_2 of Σ_2, and a marking M_3 of the three places A, B and C; moreover, these three projections determine M. Making use of this property, we write $M = (M_1, M_2, M_3)$; also, we write M_3 as a string of three numbers representing the number of tokens in A, B, and C. Finally, the null marking, i.e., the marking that puts no token in any place, is denoted by $\mathbf{0}$.

Let M_0 be the initial marking of Σ. The following lemma follows easily from the definitions.

Lemma 4.4

If $(M_{01}, M_{02}, 100) \xrightarrow{\sigma} (M_1, M_2, 001)$ is an occurrence sequence of Σ, then there exist sequences $\sigma_1, \sigma_2, \sigma_3$ such that $\sigma = \sigma_1\, t_{AB}\, \sigma_2\, t_{BC}\, \sigma_3$. ■

We can now characterise the 'yes' instances of the containment problem in terms of occurrence sequences of Σ.

Lemma 4.5

A triple (Σ_1, Σ_2, f) is a 'yes' instance of the containment problem iff every occurrence sequence $(M_{01}, M_{02}, 100) \xrightarrow{\sigma\, t_{AB}} (M_1, M_{02}, 010)$ of Σ can be extended to a maximal occurrence sequence $(M_{01}, M_{02}, 100) \xrightarrow{\sigma\, t_{AB}\, \tau} (\mathbf{0}, \mathbf{0}, 001)$.

Proof: ($\Rightarrow$): M_1 is a reachable marking of Σ_1. Since M_1 is reachable in Σ_2 as well, there exists a sequence $\sigma_2\, t_{BC}$ such that

$$(M_1, M_{02}, 010) \xrightarrow{\sigma_2\, t_{BC}} (M_1, M_1, 001)$$

Let σ_3 be a sequence which contains each transition of the form $\{s, f(s), C\} \to \{C\}$ $M(s)$ times, and no occurrence of any other transition. We then have

$$(M_1, M_1, 001) \xrightarrow{\sigma_3} (\mathbf{0}, \mathbf{0}, 001)$$

By the definition of Σ, no transition is enabled at the marking $(\mathbf{0}, \mathbf{0}, 001)$, and therefore

$$(M_0, M_0, 100) \xrightarrow{\sigma\, t_{AB}\, \sigma_2\, t_{BC}\, \sigma_3} (\mathbf{0}, \mathbf{0}, 001)$$

is maximal.

($\Leftarrow$): Let M_1 be a reachable marking of Σ_1. Then, we have

$$(M_{01}, M_{02}, 100) \xrightarrow{\sigma_1} (M_1, M_{02}, 100) \xrightarrow{t_{AB}} (M_1, M_{02}, 010)$$

for some sequence σ_1. So there exists a sequence τ such that

$$(M_{01}, M_{02}, 010) \xrightarrow{\sigma\, t_{AB}\, \tau} (\mathbf{0}, \mathbf{0}, 001)$$

is a maximal occurrence sequence. By Lemma 4.4, $\tau = \sigma_2\, t_{BC}\, \sigma_3$ for some sequences σ_2 and σ_3. Let

$$(M_1, M_{02}, 010) \xrightarrow{\sigma_2\, t_{BC}} (M_1', M_2, 010) \xrightarrow{\sigma_3} (\mathbf{0}, \mathbf{0}, 100)$$

We show $M_1' = M_1 = M_2$, which proves that M_1 is reachable in Σ_2.
Since σ_2 occurs after t_{AB} has occurred, no transition of Σ_1 occurs in σ_2. So $M_1' = M_1$.
Since σ_3 occurs after t_{BC}, only transitions of the form $\{s\} \to \{s\}$ for some place s can occur in σ_3. Since these transitions remove one token from s and one from $f(s)$, we have $M_1' - \mathbf{0} = M_2 - \mathbf{0}$. So $M_1' = M_2$. ■

To finish the undecidability proof, it suffices to encode the characterisation of Lemma 4.5 into the temporal logic defined above. Let *Dead* be the conjunction, for every transition t of Σ, of the formula $\neg EN(t)$. It follows that a marking satisfies *Dead* iff it is dead, i.e., it does not enable any transition.

Theorem 4.6

A triple (Σ_1, Σ_2, f) is a 'yes' instance of the containment problem iff Σ satisfies the formula

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

Proof: The formula of the theorem states that every occurrence sequence $\sigma_1\, t_{AB}$ of Σ can be extended to an occurrence sequence $\sigma_1\, t_{AB}\, \tau$ leading to a dead marking. Since for every place s of Σ, with the exception of C, there exists a transition $\{s\} \to \{s\}$, this dead marking can only be $(\mathbf{0}, \mathbf{0}, 001)$. Apply then Lemma 4.5. ■

4.7 Decidability of UB$^-$ for BPPs and other Petri net classes

We prove the decidability of the model checking problem for UB$^-$ and Petri nets whose set of reachable markings 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 ith component of the vector indicates the number of tokens on the ith place, for some fixed order of the places. We can then identify the set of reachable markings with a set of vectors.

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

$$\mathcal{X} = \{v + a_1 v_1 + \ldots + a_k v_k \mid a_1, \ldots, a_k \geq 0\}$$

$\mathcal{X}$ is *semilinear* if it is a finite union of linear sets.

To prove decidability, we reduce the problem to the validity problem of Presburger arithmetic, the first order theory of addition, which is known to be decidable (see, for instance, [2]).

A classical result of Ginsburg and Spanier [17] states that every semilinear set can be expressed in Presburger arithmetic. We use it as follows. Let $\Sigma = (N, M_0)$ be a Petri net whose set of reachable markings is semilinear. Let $\mathbf{M_0}$ be the vector representation of M_0, and let $\mathbf{M}$ be a vector of free variables, one for each place of N. Then there exists a formula $Reach_{\mathbf{M_0}}(\mathbf{M})$ of Presburger arithmetic, such that a marking M is reachable from M_0 if and only if the closed formula obtained substituting M for $\mathbf{M}$ in $Reach_{\mathbf{M_0}}$ is valid.

Using this result, we prove the following:

Theorem 4.7

Let Σ be a net whose set of reachable markings is effectively semilinear. It is decidable if Σ satisfies a formula of UB$^-$.

Proof: Given a formula ϕ of UB$^-$, we prove that the set of reachable markings that satisfy it can be expressed by a formula $F_\phi(\mathbf{M})$ of Presburger arithmetic. The proof is by structural induction on the syntax of UB$^-$:

- $\phi = \mathbf{true}$. The set of reachable markings that satisfy ϕ is the set of all reachable markings of Σ, which is semilinear by hypotesis. So $F_{\mathbf{true}}(\mathbf{M}) = \mathit{Reach}_{\mathbf{M_0}}(\mathbf{M})$.
- $\phi = \neg\psi$. 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 eeasily 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

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

  encodes the markings which enable t and, once t is fired, are transformed into markings satisfying ψ.
- $\phi = EF\psi$. Then we take

  $$F_{EF\psi}(\mathbf{M}) = \exists \mathbf{M}' \; (\; \mathit{Reach}_{\mathbf{M}}(\mathbf{M}') \; \wedge \; F_\psi(\mathbf{M}') \;)$$

Now, to check if N satisfies a formula ϕ, it suffices to decide if the formula $F_\psi(\mathbf{M_0})$ is valid. ∎

It was proved in [13] that the sets of reachable markings of the Petri nets corresponding to BPPs in normal form are effectively semilinear. There exist several other classes of Petri nets with effectively semilinear sets of markings, in particular persistent and conflict-free Petri nets [15].

The following proposition shows that the model checking problem for UB$^-$ and BPPs is PSPACE-hard, even if the BPPs are finite state. Using results of [13], it is easy to show that the model checking problem for formulae without nested temporal operators (and arbitrary BPPs) is NP-complete. We conjecture that the problem for formulae 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 formulae is PSPACE-complete, which would yield an exponential-time upper bound. Since it is known that checking validity in Pressburger arithmetic requires at least doubly exponential nondeterministic time [16], this result would improve the upper bound derived from Proposition 4.7.

Theorem 4.8

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

Proof: The proof is a routine reduction from the problem Quantified Boolean Formulae. 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 $\mathtt{S}$ and a formula $True(\Phi)$ of UB^- such that Φ is true iff $\mathtt{S}$ satisfies $True(\Phi)$.

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

$$\mathtt{True(C_j)} \stackrel{\mathrm{def}}{=} \mathtt{true(C_j)} \cdot \mathbf{0}$$

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

$$\begin{array}{rcl} \mathtt{X_i} & \stackrel{\mathrm{def}}{=} & \mathtt{true(X_i)} \cdot \mathtt{True(X_i)} + \mathtt{false(X_i)} \cdot \mathtt{False(X_i)} \\ \mathtt{True(X_i)} & \stackrel{\mathrm{def}}{=} & \mathtt{prop(X_i)} \cdot (\, \|_{\,C_j \in \mathcal{C}_i^t} \mathtt{True(C_j)}) \\ \mathtt{False(X_i)} & \stackrel{\mathrm{def}}{=} & \mathtt{prop(X_i)} \cdot (\mathtt{false(X_i)}. \, \|_{\,C_k \in \mathcal{C}_i^f} \mathtt{True(C_k)}) \end{array}$$

The process $\mathtt{X_i}$ can perform either a $\mathtt{true(X_i)}$ or a $\mathtt{false(X_i)}$ 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:

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

Since none of the equations of this BPP is recursive, $\mathtt{S}$ is a finite state process.

Let us now construct the formula $True(\Phi)$. First, we define for every $0 \leq i \leq n$ the following auxiliary formula

$$\begin{array}{rcl} Assigned(i) & = & EN(\mathtt{prop(X_1)}) \wedge \ldots \wedge EN(\mathtt{prop(X_i)}) \wedge \\ & & EN(\mathtt{true(X_{i+1})}) \wedge \ldots EN(\mathtt{true(X_n)}) \end{array}$$

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 `true` actions would no longer be enabled.

We define the formula $True(\Phi)$ as follows:

$EF(Assigned(1) \wedge$
$\quad AG(Assigned(2) \Rightarrow$
$\quad\quad EF(Assigned(3) \wedge$

$\ldots$

$$EF(Assigned(n) \wedge$$
$$EN(\mathtt{true(C_1)} \wedge \ldots \wedge EN(\mathtt{true(C_n)}))\ldots)))$$

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. ■

5 Adding atomic sentences

5.1 Linear Time logics

It is well known that Petri nets have less computing power than Turing machines. Crudely speaking, the reason is that Petri nets cannot 'test for zero'. This means that it is not possible in general to add new places and transitions to a given Petri net having a distinguished place s, in such a way that (1) the behaviour of the old net is not disturbed, and (2) one of the new transitions is enabled exactly when s is *not* marked. If Petri nets are extended with inhibitor arcs, which allow to 'test for zero', they become Turing powerful [30].

Instead of extending Petri nets, we will add atomic sentences to a simple linear time temporal logic (easily embeddable into the linear time μ-calculus), which allow it to 'test for zero'. These atomic sentences have the form $s = 0$, and they are given the semantics 'the place s contains no tokens at the current marking'. We shall construct a formula of this logic which holds for the second weak model of a counter machine if and only if the formula *Halt* of UB holds. This proves that the model checking problem for this logic and VBPPs is undecidable.

The syntax of the logic is

$$\phi ::= \mathbf{true} \mid s = 0 \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid (a)\phi \mid G\phi$$

where s is a place of the Petri net $\Sigma = (N, M_0)$ on which the formulae are to be interpreted.

The semantics is as for the linear time mu-calculus, with the addition of

$$\begin{array}{rcl} \|s = 0\|_{\mathcal{V}} & = & \{\sigma \in L((N, M)) \mid M(s) = 0\} \\ \|G\phi\|_{\mathcal{V}} & = & \{\sigma \in Act^{\infty} \mid \tau \in \|\phi\|_{\mathcal{V}} \text{ for every suffix } \tau \text{ of } \sigma\} \end{array}$$

So a sequence satisfies $s = 0$ if it can occur from some marking of N that puts no tokens on s.

To see that this logic can be embedded in the linear time mu-calculus, just notice that the formula $\nu X.\phi \wedge \bigvee_{a \in Act}(a)X$ has the same denotation as $G\phi$.

As in the case of the linear time mu-calculus, we say that Σ satisfies a formula ϕ if every maximal occurrence sequence of Σ satisfies ϕ.

We define a formula ϕ'_h of this logic as follows: replace each occurrence of an expression $EN(a)$ in the formula ϕ_h of UB by the formula

$$EN'(a) = \bigvee_{\{t \| l(t) = a\}} \bigwedge_{s \in {}^\bullet t} \neg\, s = 0$$

This formula expresses that all the input places of some transition labelled by a are marked. So $EN'(a)$ holds for a sequence of $L((N, M))$ if M enables the action a.

Therefore, a sequence of $L((N, M))$ satisfies ϕ'_h iff the marking M satisfies the formula ϕ_h of UB.

We now construct the formula

$$Halt' = G(\ \neg\phi_h\ \vee\ EN'(\texttt{halt})\ )$$

Σ satisfies *Halt'* if for all its maximal occurrence sequences, every marking reached along the execution of σ enables the transition labeled by `halt`, or does not satisfy ϕ_h. This is the case if and only if Σ satisfies the formula *Halt* of UB. It follows that the model checking problem for this temporal logic and VBPPs is undecidable.

5.2 Branching Time Logics

.

Theorem 4.7 has been proved in Section 3 for a version of the logic UB^- having only **true** as atomic sentence. It follows from the proof of the theorem that any atomic sentence expressible in Presburger arithmetic can be added to this logic without spoiling the decidability result. 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 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, [13] not only proves the semilinearity of the set of reachable markings, but also the semilinearity of the set of *Parikh mappings* of the occurrence sequences. For a net with n transitions, the Parikh mapping of an occurrence sequence is a vector of dimension n which represents how many times each transition occurs in the sequence. This result allows us to extend UB^- 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 can be reached from M by the execution of an occurrence sequence 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 BPPs, two concurrent models with very different expressive powers, and several logics.

Our first conclusion is the existence of a decidability gap between linear time and branching time logics. While the linear time mu-calculus is decidable for a powerful model like Petri nets, a not very expressive branching time logic like UB is undecidable for a very weak model like VBPPs.

This gap does not exist for sequential models like context-free processes or pushdown automata. For them, the modal mu-calculus, which (strictly) contains the linear time mu-calculus and UB, is decidable.

The addition of simple state-based atomic sentences to linear time logics makes them as undecidable as branching time logics. These sentences provide information about which places are marked, and this information determines which transitions are enabled. 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 express which are the possible alternatives for the next step.

Finally, we have proved that the fragment UB^- of UB is still undecidable for arbitrary Petri nets, but decidable for nets with semilinear sets of reachable markings, which include BPPs and persistent systems.

References

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

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

[3] J. C. Bradfield: Verifying Temporal Properties of Systems. Birkhäuser, Boston, Massachussets ISBN 0-8176-3625-0 (1991).

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

[5] O. Burkart and B. Steffen. Pushdown Processes: Parallel Composition and Model Checking. To appear in the proceedings of CONCUR '94 (1994).

[6] S. Christensen. Ph. D. Thesis, University of Edinburgh (1993).

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

[8] 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).

[9] 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).

[10] 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).

[11] E.A. Emerson and Srinivasan. Branching Time Temporal Logic. LNCS 354, 123–172 (198?).

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

[13] J. Esparza. On the uniform word problem for commutative context-free grammars. Submitted for publication (1994).

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

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

[16] 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)

[17] S. Ginsburg and E.H. Spanier. Semigroups, Presburger formulas and languages. Pacific Journal of Mathematics 16, 285–296.

[18] M.H.T. Hack: Decidability questions for Petri nets. Ph. D. Thesis, MIT (1976).

[19] Y. Hirshfeld. Petri Nets and the Equivalence Problem. Proceedings of CSL '93 (1994).

[20] 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).

[21] 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).

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

[23] H. Hungar. Model Checking of Higher-Order Processes. To appear in the proceedings of CONCUR '94. (1994).

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

[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. To appear in the Proceedings of CONCUR '94 (1994).

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

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

[29] D. Muller and P. Schupp. The Theory of Ends, Pushdown Automata and Second Order Logic. Theoret. Comput. Sci. 37, 51–75 (1985).

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

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

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

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