On the model checking problem for branching time logics and Basic Parallel Processes

Javier Esparza and Astrid Kiehn
Institut für Informatik, Technische Universität München
Arcisstr.21, D–80290 München, Fax +49 2105 8207
{esparza,kiehn}@informatik.tu-muenchen.de

Abstract

We investigate the model checking problem for branching time logics and Basic Parallel Processes. We show that the problem is undecidable for the logic $\forall L(O, F, U)$ (equivalent to CTL^*) in the usual interleaving semantics, but decidable in a standard partial order interpretation.

1 Introduction

Most techniques for the verification of concurrent systems are only applicable to the finite state case. However, many interesting systems have infinite state spaces. In the last years, several verification problems have been shown to be decidable for two classes of infinite-state systems, namely the processes of Basic Process Algebra (BPA) [1], a natural subset of ACP, and the Basic Parallel Processes (BPP) [3], a natural subset of CCS. These results can be classified into those showing the decidability of equivalence relations [3, 4], and those showing the decidability of model checking for different modal and temporal logics. In this paper, we contribute to this second group. In the sequel, when we say that a logic is decidable for a class of processes, we mean that the model checking problem is decidable.

BPA processes are recursive expressions built out of actions, variables, and the operators sequential composition and choice. They are a model of sequential computation. For BPA processes, the modal mu-calculus, the most powerful of the modal and temporal logics commonly used for verification, is known to be decidable. The proof is a complicated reduction to the validity problem for S2S (monadic second order logic of two successors) [15, 8]. Simpler algorithms have been given for the alternation-free fragment of the mu-calculus [2, 14].

BPPs are recursive expressions built out of actions, variables, and the operators prefix, choice, and parallel composition. BPPs without the parallel operator have the same expressive power as finite automata. Therefore, they are a sort of minimal concurrent extension of finite automta, and so a good starting point for the study of concurrent infinite-state systems. In [10] it was shown that the linear time mu-calculus, which contains many other linear time logics, like PLTL [6] or EL [22] is decidable. It was also shown that the modal mu-calculus is undecidable. The decidability of branching time logics like CTL [5], or CTL*[7], which are some of the most frequently used for automatic verification in the finite-state case, was open.

In this contribution, we consider a logic equivalent to CTL* and two interpretations: the usual one based on the interleaving of concurrent actions, and a natural partial order interpretation.

In the first half of the paper, we prove that, in the interleaving interpretation, a small fragment of this logic (equivalent to the fragment of CTL formed by propositional logic, AX, and AF), is already undecidable for BPPs, even for BPPs without the choice operator. Since a result of [11] shows that the fragment containing AG instead of AF is decidable, this establishes the decidability border for branching time logics in this interpretation.

In the second half of the paper, we prove that the logic is decidable in the partial order interpretation (more precisely, we prove it for a subset of BPPs, and show how our results could be extended to the whole class).

The paper is organised as follows. Section 2 introduces Basic Parallel Processes. Section 3 describes the syntax and interleaving semantics of the logic

$\forall L(O, F, U)$. The undecidability result for the interleaving interpretation is contained in Section 4. Section 5 gives a Petri net semantics for a subclass of BPPs. Using this semantics, Section 6 gives a partial order interpretation of $\forall L(O, F, U)$. The decidability of model checking for this interpretation is contained in Section 7.

2 Basic and Very Basic Parallel Processes

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

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

where a belongs either to a set of *atomic actions* Act. 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

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

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*.

Any BPP determines a labelled transition system $\mathcal{T} = (\mathcal{S}, \{\stackrel{a}{\longrightarrow} \mid a \in Act\})$, whose states are the BPP expressions reachable from the leading variable, and whose transition relations are the least relations satisfying the following rules:

$$a \cdot E \stackrel{a}{\longrightarrow} E \qquad \frac{E \stackrel{a}{\longrightarrow} E'}{X \stackrel{a}{\longrightarrow} E'} \, (X \stackrel{\text{def}}{=} E)$$

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

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

3 The logic $\forall L(O, F, U)$

Stirling uses in [20] the notation $L(Op_1, \ldots, Op_n)$ to name the linear-time temporal language whose temporal operators are $Op_1, \ldots, Op_n$. He also uses $\forall L(Op_1, \ldots, Op_n)$ to name the language obtained by extending $L(Op_1, \ldots, Op_n)$ with the branching operator $\forall$, which allows to quantify on paths. We stick to this notation, with a small deviation, namely that the logics we consider have **true** as only atomic proposition, instead of a set of propositional variables.

The syntax of $\forall L(O, F, U)$ with a sort of labels $\mathcal{L}$ is given by the following grammar:

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

where $a \in \mathcal{L}$. $\exists$ abbreviates $\neg\forall\neg$.

Let $\mathcal{T}$ be the transition system of a BPP E over Act. We interpret $\forall L(O, F, U)$ with sort of labels Act on $\mathcal{T}$. We need some preliminary definitions. A *path* of $\mathcal{T}$ is a (finite or infinite) sequence $s_0 \stackrel{a_0}{\longrightarrow} s_1 \stackrel{a_1}{\longrightarrow} \ldots$ of states s_i and labels a_i. A path π is a *run* if it is maximal, i.e. either it is infinite or it is finite of length n and there is no a, s such that $s_n \stackrel{a}{\longrightarrow} s$. Given a run π, $Min(\pi)$ denotes s_0. Given two runs π and π', we say $\pi \sqsubseteq \pi'$ if π' is a suffix of π, and we say $\pi \stackrel{a_0}{\longrightarrow} \pi'$ if $\pi = s_0 \stackrel{a_0}{\longrightarrow} s_1 \stackrel{a_1}{\longrightarrow} \ldots$ and $\pi' = s_1 \stackrel{a_1}{\longrightarrow} \ldots$.

The denotation of a formula is a set of runs through $\mathcal{T}$, defined according to the following rules:

$$
\begin{array}{rcl}
\|\neg\phi\| & = & \mathcal{R} - \|\phi\| \\
\|\phi_1 \wedge \phi_2\| & = & \|\phi_1\| \cap \|\phi_2\| \\
\|\forall\phi\| & = & \{\pi \in \mathcal{R} \mid \forall\pi' \in \mathcal{R} \, . \\
& & \quad Min(\pi') = Min(\pi) \Rightarrow \pi' \in \|\phi\|\} \\
\|(a)\phi\| & = & \{\pi \in \mathcal{R} \mid \pi \stackrel{a}{\longrightarrow} \pi' \wedge \pi' \in \|\phi\|\} \\
\|F\phi\| & = & \{\pi \in \mathcal{R} \mid \exists\pi' \in \mathcal{R} \, . \, \pi \sqsubseteq \pi' \wedge \pi' \in \|\phi\|\} \\
\|\phi_1 U \phi_2\| & = & \{\pi \in \mathcal{R} \mid \exists\pi' \in \mathcal{R} \, . \, \pi \sqsubseteq \pi' \wedge \pi' \in \|\phi_2\| \\
& & \wedge \ \forall\pi'' \in \mathcal{R} \, . \, \pi \sqsubseteq \pi'' \sqsubset \pi' \Rightarrow \pi'' \in \|\phi_1\|\}
\end{array}
$$

where $\mathcal{R}$ denotes the set of runs of $\mathcal{T}$.

Observe that the operator $\forall$ is a quantifier over all paths starting at a particular state.

We say that $\mathcal{E}$ satisfies a formula ϕ if

$$\forall\pi \in \mathcal{R} \, . \, Min(\pi) = X_1 \Rightarrow \pi \in \|\phi\|$$

where X_1 is the leading variable of $\mathcal{E}$.

In the sequel we refer to these definitions as the *interleaving interpretation* of $\forall L(O, F, U)$.

4 Undecidability of the interleaving interpretation

We show in this section that the model checking problem for the language $\forall L(O,F,U)$ and BPPs is undecidable under the interleaving interpretation. In fact, we show that the problem is already undecidable for VBPPs and the following sublanguage of $\forall L(O,F,U)$:

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

Notice that this is a pure branching-time language, because the linear time operators O and F can only appear quantified. Following [20], we call it $B(O,F)$.

Branching-time logics have another interpretation, equivalent to the one given above, in which the denotation of a formula is a set of states. A state belongs to the new denotation of a formula iff all the runs starting at it belong to the old denotation. We use this interpretation in this section.

We prove undecidability by a reduction from the halting problem of counter machines whose counters are initialised to 0 [16].

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 $q_0,\ldots,q_n$ are of 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. The *initial configuration* is $(q_0,0,\ldots,0)$. 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. It is undecidable whether a counter machine halts [16].

Given a counter machine $\mathcal{M}$, our reduction constructs a VBPP with leading variable `M`, and a formula *Halt* of $B(O,F)$ such that $\mathcal{M}$ halts if and only if the VBPP 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

$$\mathit{Halt} = \forall F\, \exists(\mathtt{halt})\mathbf{true}$$

which expresses that `M` eventually reaches a state from which it can do `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 from `M`, 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 shall replace the formula *Halt* above by another one, more complicated. First, we shall construct a formula ϕ_h satisfying the following two properties:

(1) there exists a run starting at the leading variable whose states satisfy ϕ_h, and

(2) if all the states of a run starting at the leading variable satisfy ϕ_h, then the run is honest.

Then, we shall define

$$\mathit{Halt} = \forall F(\ \neg\phi_h\ \vee\ \exists(\mathtt{halt})\,\mathbf{true}\ )$$

If the model `M` of the counter machine satisfies *Halt*, then the runs starting at `M` that satisfy ϕ_h at every state must contain a state satisfying $\exists(\mathtt{halt})\,\mathbf{true}$. 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 starting at `M` either is honest or contains a state which does not satisfy ϕ_h. In the first case, since the machine terminates, the run contains a state satisfying $\exists(\mathtt{halt})\,\mathbf{true}$, and therefore it satisfies *Halt*. In the second case, the run directly satisfies *Halt*.

We construct the VBPP model in two steps. First, we describe a rather straightforward VBPP model. Unfortunately, it is not possible to find the formula ϕ_h for it. We solve this problem by 'refining' this model in an appropiate way.

A first '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 transition rule. A state q_i of type I is modelled by

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

A state q_i of type II with is modelled by

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

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 describe the choice between q_k and q'_k we need the choice operator.

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

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

Finally, $\mathtt{M}$ is defined by

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

It follows easily from the operational semantics of BPPs that the reachable states of $\mathtt{M}$ 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 $\underbrace{\mathtt{P} \parallel \ldots \parallel \mathtt{P}}_{k}$ (and $\mathtt{P}^0$ means that the state contains no copies of $\mathtt{P}$ at all). The reachable states 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 the state, and the indices $j_1, \ldots, j_m$ the values of the counters. We say that these states are *meaningful.*

The *honest* runs of $\mathtt{M}$ are defined 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 runs, but not every run of $\mathtt{M}$ is honest.

A second 'weak' model. Following an idea introduced by Hirshfeld in [13], we split the actions of the first model. A counter c_j is now modelled by

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

A state q_i of type II is modelled by

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

In the other equations we replace $\mathtt{in_i}$ and $\mathtt{out_i}$ by $\mathtt{in^1_i}$ and $\mathtt{out^1_i}$ for consistency, but the actions are not splitted.

In order to describe the formula ϕ_h, we first introduce some notations. Define

$$EN(a_1, \ldots, a_k) \equiv \bigwedge_{i=1}^{k} \exists(a_i)\,\mathbf{true}$$

where EN stands for ENabled. Now, let A be the set of actions of the form $\mathtt{out^1_i}$, $\mathtt{out^2_i}$, $\mathtt{dec^2_i}$ or $\mathtt{dec^3_i}$, and let $a_1, \ldots, a_k$ be actions of A. Define

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

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 $\widehat{EN}(\mathtt{out}^1_\mathtt{i})$. For each state q_i of type II, ϕ_h contains two disjuncts. The first is

$$\neg EN(\mathtt{dec}^1_\mathtt{j}) \wedge \neg EN(\mathtt{dec}^2_\mathtt{j}) \wedge \neg EN(\mathtt{dec}^3_\mathtt{j}) \ \wedge$$

$$(\widehat{EN}(\mathtt{out}^1_\mathtt{i}) \ \vee \ \widehat{EN}(\mathtt{out}^2_\mathtt{i}) \vee$$

$$\widehat{EN}(\mathtt{out}^2_\mathtt{i}, \mathtt{out}^1_\mathtt{k}) \ \vee \ \widehat{EN}(\mathtt{out}^1_\mathtt{k}))$$

and the second is

$$(EN(\mathtt{dec}^1_\mathtt{j}) \vee EN(\mathtt{dec}^2_\mathtt{j}) \vee EN(\mathtt{dec}^3_\mathtt{j})) \ \wedge$$

$$(\widehat{EN}(\mathtt{out}^1_\mathtt{i}) \ \vee \ \widehat{EN}(\mathtt{out}^1_\mathtt{i}, \mathtt{dec}^2_\mathtt{j}) \vee$$

$$\widehat{EN}(\mathtt{out}^2_\mathtt{i}, \mathtt{dec}^2_\mathtt{j}) \ \vee \ \widehat{EN}(\mathtt{out}^2_\mathtt{i}, \mathtt{dec}^3_\mathtt{j}) \vee$$

$$\widehat{EN}(\mathtt{out}^2_\mathtt{i}, \mathtt{out}^1_{\mathtt{k}'}, \mathtt{dec}^3_\mathtt{j}) \ \vee \ \widehat{EN}(\mathtt{out}^1_{\mathtt{k}'}, \mathtt{dec}^3_\mathtt{j}))$$

It is easy to see that some run starting at $\mathtt{M}$ satisfies ϕ_h. The following lemma proves that ϕ_h also satisfies condition (2).

Lemma 4.1 *If all the states of a run of* $\mathtt{M}$ *satisfy the formula* ϕ_h*, then the run is honest.*

Proof Consider an arbitrary meaningful state

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

of a run in which every state satisfies ϕ_h. 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. 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; (2) q_i is of type II, and $i_j = 0$; (3) q_i is of type II, and $i_j > 0$. 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 $\mathtt{out}^1_\mathtt{i}$, 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 $\mathtt{out}^1_\mathtt{i}$ and some other $\mathtt{out}$ action are enabled, or the sequence $\mathtt{out}^1_\mathtt{i}\ \mathtt{out}^1_\mathtt{i}$ is enabled. Such a state does not satisfy ϕ_h. The $\mathtt{dec}^1$ actions lead to states where some $\mathtt{dec}^2$ action, different from $\mathtt{dec}^2_\mathtt{j}$, is enabled. Since the only enabled $\mathtt{out}^1$ action is $\mathtt{out}^1_\mathtt{i}$, they states do not satisfy ϕ_h either. So the next action in the run can only be $\mathtt{out}^1_\mathtt{i}$. Then we have $E \xrightarrow{\mathtt{out}^1_\mathtt{i}} E'$, where

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

At the new state E', the enabled actions 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 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 $\mathtt{in}^1_\mathtt{k}$, which leads to the state

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

The actions enabled at E'' are $\mathtt{out}^1_\mathtt{k}$, $\mathtt{out}^2_\mathtt{i}$ and the $\mathtt{dec}^1$ actions of the nonempty counters. It is easy to see that the only next possible action is $\mathtt{out}^2_\mathtt{i}$, 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 run executes $\mathtt{out}^1_\mathtt{i}\ \mathtt{in}^1_\mathtt{k}\ \mathtt{out}^2_\mathtt{i}$ from E. 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 $\mathtt{out}^1_\mathtt{i}$, and in case (3) the only possible sequence is

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

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

Now, we 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*.

Theorem 4.2 *The model checking problem for the logic* $B(O, U)$ *and VBPPs is undecidable.*

5 A partial order interpretation of $\forall L(O,F,U)$

We give a partial order interpretation of $\forall L(O,F,U)$ for the subclass of simple BPPs. More precisely, we translate simple BPPs into Petri nets, and then use the standard partial order semantics of Petri nets given in [9].

The subclass of *simple* BPP expressions is defined in two steps:

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

$$\begin{array}{rcll} E & ::= & S & \text{(an initially sequential process)} \\ & | & E \parallel E & \text{(merge)} \end{array}$$

In general, simple BPP processes are not finite-state but they can be characterised using a finite set of process expressions. To a family of recursive equations $\mathcal{E} = \{X_i \stackrel{\text{def}}{=} E_i \mid 1 \leq i \leq n\}$ we associate the set of generators $Gen(\mathcal{E}) = \bigcup Gen(E_i)$ defined by:

$$\begin{array}{rcl} Gen(X) & = & \emptyset \\ Gen(\mathbf{0}) & = & \{\mathbf{0}\} \\ Gen(a \cdot E_1) & = & \{a \cdot E_1\} \cup Gen(E_1) \\ Gen(E_1 + E_2) & = & \{E_1 + E_2\} \cup (Gen(E_1) \setminus \{E_1\}) \\ & & \cup (Gen(E_2) \setminus \{E_2\}) \\ Gen(E_1 \parallel E_2) & = & Gen(E_1) \cup Gen(E_2) \end{array}$$

Note that all generators in $Gen(E)$ are initially sequential.

Let $\equiv$ denote the congruence generated by the equations expressing commutativity and associativity of $\parallel$. We use $\prod_{i \in I} S_i$ to denote the parallel product $S_{i_1} \parallel S_{i_2} \parallel \ldots \parallel S_{i_k}$ where I is the finite index set $\{i_1, \ldots i_k\}$. Given an expression $E \equiv \prod_{i \in I} S_i$, let $|E|$ be the multiset of parallel components of $\prod_{i \in I} S_i$. The number of occurences of an element G in the multiset $|E|$ is denoted by $|E|_G$.

Proposition 5.1 *Let $\mathcal{E} = \{X_i \stackrel{\text{def}}{=} E_i \mid 1 \leq i \leq n\}$ be a simple BPP process.*

1. *$Gen(\mathcal{E})$ is finite,*
2. *$E_i \equiv \prod_{j \in J} S_j$ for a finite index set J and $S_j \in Gen(\mathcal{E})$,*
3. *if $G \in Gen(\mathcal{E})$ and $G \xrightarrow{\mu} H$ then there are finite index sets J and K such that $H \equiv \prod_{j \in J} S_j \parallel \prod_{k \in K} X_{i_k}$ where all $S_j \in Gen(\mathcal{E})$ and all X_{i_k}'s are variables of $\mathcal{E}$,*
4. *for each $G \in Gen(\mathcal{E})$ and each $G \xrightarrow{\mu} H$ there is exactly one representation according to 3. (up to $\equiv$).*

5.1 The Net Representation

A *labelled net* is a fourtuple (S, T, W, l), where S, T are disjoint sets of *places* and *transitions*, $W\colon (S \times T) \cup (T \times S) \to \mathbb{N}$ is a *weight* function, and $l\colon T \to \mathcal{L}$ is a *labelling* function. For $x \in S \cup T$, ${}^\bullet x = \{y \in S \cup T \mid W(y,x) > 0\}$ and $x^\bullet = \{y \in S \cup T \mid W(x,y) > 0\}$. A *marking* of a net is a function $M\colon S \to \mathbb{N}$. A *Petri net* is a pair (N, M_0), where N is a net and M_0 is a marking of N, called the *initial marking.*

The net of a simple BPP process is obtained by taking its generators as places. The transitions a generator can perform determine the Petri net transitions. If $G \xrightarrow{\mu} H$ for a generator G, then the net contains a transition with G as input place. The definition of the output places is a bit more involved, because the process H is not necessarily a generator. Due to the previous proposition we know that H can be uniquely represented (up to $\equiv$) as a parallel product of generators and variables. Moreover, the variables are defined by expressions which, due to their guardedness, can also be seen as a parallel product of generators. In this way we can uniquely associate to H a multiset of generators. The elements of this multiset are the output places of the transition.

$N(\mathcal{E}) := (S_\mathcal{E}, T_\mathcal{E}, W_\mathcal{E}, l_\mathcal{E})$ where

$$\begin{array}{rcl} S_\mathcal{E} & = & Gen(\mathcal{E}) \\ T_\mathcal{E} & = & \{(G, \mu, H) \mid G \in Gen(\mathcal{E}), G \xrightarrow{a} H\} \\ W_\mathcal{E}(G,t) & = & \begin{cases} 1 & \text{if } t = (G,a,H) \\ 0 & \text{otherwise} \end{cases} \\ W_\mathcal{E}(t,G) & = & |\prod_{j \in J} S_j|_G + \sum_{kinK} |E_{i_k}|_G \text{ with} \\ & & t = (G', a, H), H \equiv \prod_{j \in J} S_j \parallel \prod_{k \in K} X_{i_k} \\ l_\mathcal{E}(t) & = & a \text{ where } t = (G,a,H) \end{array}$$

The initial marking $M_0^\mathcal{E}$ is defined by $M_0^\mathcal{E}(G) = |E_1|_G$ for every $G \in Gen(\mathcal{E})$ where E_1 is the expression defining the leading variable X_1. So the Petri

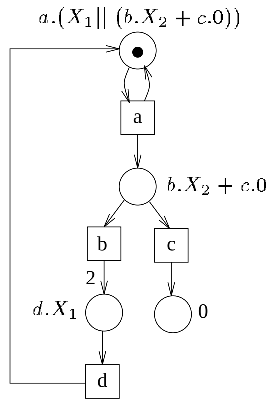

Figure 1: Net representation of the BPP given in the text.

net associated to $\mathcal{E}$ is

$$PN(\mathcal{E}) = (N(\mathcal{E}), M_0^{\mathcal{E}}).$$

The net representation of

$$X_1 \stackrel{\text{def}}{=} a \cdot (X_1 \parallel (b \cdot X_2 + c \cdot \mathbf{0}))$$
$$X_2 \stackrel{\text{def}}{=} d \cdot X_1 \parallel d \cdot X_1$$

with leading variable X_1 is given in Figure 1. Only the names of the places and the labels of the transitions are shown. The main property of the net representation follows immediately from the definitions:

Proposition 5.2 *In the net representation of a BPP process, every transition t has exactly one input place s, and the weight of the arc from s to t is 1.*

6 The unfolding of a Petri net

In this section we define partial order counterparts of the notions of labelled transition system, state, and run, that were used in the interleaving interpretation.

Unfoldings. The counterpart of a labelled transition system is the *unfolding* of the Petri net, a well known partial order semantics [9]. The unfolding of a Petri net is an acyclic net, usually infinite. Figure 2 shows an initial part of the infinite unfolding of the Petri net shown in Figure 1. Although the notion of unfolding is intuitively rather clear, its formal definition requires some effort. We follow the lines of [9], with a small change: in [9], unfoldings are defined for nets (S, T, W) in which the weights $W(x, y)$ have the value 0 or 1 for every two nodes x and y. We generalise them to *labelled* nets with *arbitrary* weights.

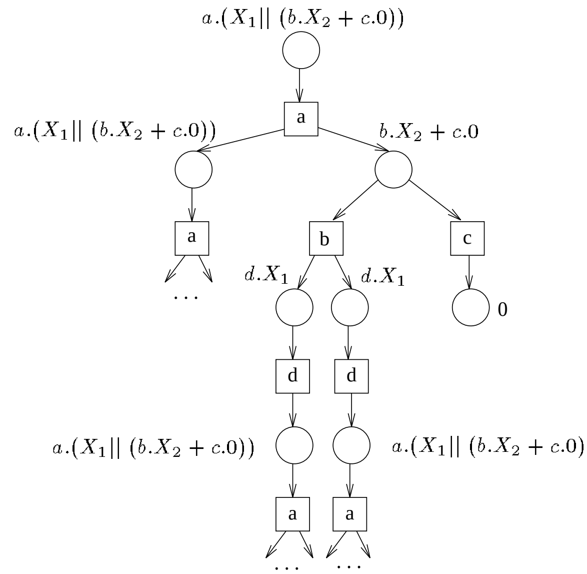

Figure 2: The unfolding of the Petri net of Figure 1

Let (S, T, W) be a net and let $x_1, x_2 \in S \cup T$. The nodes x_1 and x_2 are in *conflict*, denoted by $x_1 \# x_2$, if there exist distinct transitions $t_1, t_2 \in T$ such that ${}^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset$, and there exist paths in the net leading from t_1 to x_1, and from t_2 to x_2. For $x \in S \cup T$, x is in *self-conflict* if $x \# x$.

An *occurrence net* is a non-labelled net $N = (B, E, W)$ such that:

(1) the range of W is included in $\{0, 1\}$,

(2) for every $b \in B$, $|{}^\bullet b| \leq 1$,

(3) N contains no cycles,

(4) N is finitely preceded, i.e., for every $x \in B \cup E$, the set of elements $y \in B \cup E$ such that there exists a path from y to x is finite, and

(5) no $e \in E$ is in self-conflict.

The net of Figure 2 is an occurrence net.

The elements of B and E are called *conditions* and *events*, respectively. Given two nodes x, y of an occurrence net, we say $x \preceq y$ if there is a path from x to y. Due to (3), the relation $\preceq$ is a partial order. $Min(N)$ denotes the set of minimal elements of $B \cup E$ with respect to $\preceq$.

A *branching process* of a labelled Petri net $\Sigma = (N, M_0)$ is a pair $\beta = (N', p)$, where N' is a labelled occurrence net, and p is a function $p: B \cup E \to S \cup T$, satisfying the following conditions:

(i) $p(B) \subseteq S$ and $p(E) \subseteq T$,

(ii) for every place s of N, $Min(N') \cap |p^{-1}(s)| = M_0(s)$,

(iii) for every transition t and every place s of N, if an event $e \in E$ satisfies $p(e) = t$, then $|{}^\bullet e \cap p^{-1}(s)| = W(s,t)$, and $|e^\bullet \cap p^{-1}(s)| = W(t,s)$;

(iv) for every $e_1, e_2 \in E$, if ${}^\bullet e_1 = {}^\bullet e_2$ and $p(e_1) = p(e_2)$ then $e_1 = e_2$.

(v) if $p(x)$ is labelled by l, then x is also labelled by l.

In Figure 2 we show the names of the places associated to the conditions, and the labels of the transitions associated to the events.

Engelfriet proves in [9] that a Petri net has a unique maximal branching process up to isomorphism. We call this maximal element the *unfolding* of the Petri net.

We fix in the sequel a BPP $\mathcal{E}$, and the unfolding $\beta_u = (N_u, p_u)$ of the Petri net $PN(\mathcal{E})$.

Cuts. We define cuts, which are the partial order counterparts of the states of the transition system, and a relation between cuts which corresponds to the reachability relation between states.

A set B' of conditions of N_u is a *co-set* if

$$\forall b, b' \in B' : \neg(b \prec b') \wedge \neg(b' \prec b) \wedge \neg(b \# b')$$

A maximal co-set B' with respect to set inclusion is called a *cut*.

We define the following relation between cuts:

$$c_1 \sqsubseteq c_2 \quad \text{iff} \quad \forall b_1 \in c_1 \; \exists b_2 \in c_2 : b_1 \preceq b_2$$

It is easy to see that $\sqsubseteq$ is a partial order.

We define a mapping *Mark* which associates to a cut of β_u a marking of the net N.

$$Mark(c)(s) = |c \cap p_u^{-1}(s)|$$

That is, the number of tokens that $Mark(c)$ puts in the place s is equal to the number of conditions of c labelled by s.

The following proposition can be easily proved:

Proposition 6.1 *1. A marking M of N is reachable from M_0 iff there exists a cut c of β_u such that $M = Mark(c)$.*

2. Let M_1 and M_2 be two reachable markings of Σ. M_1 is reachable from M_2 iff there exist two cuts c_1 and c_2 of β_u such that $M_1 = Mark(c_1)$, $M_2 = Mark(c_2)$, and $c_1 \sqsubseteq c_2$.

(Partial order) runs. Finally, in order to give a partial order interpretation to $\forall L(O, F, U)$, we redefine the notion of run in partial order terms. To exhibit the analogy with the interleaving case, we also denote these new runs with the symbol π.

A *(partial order) run* of β_u is a pair $\pi = (N, p)$, where N is a subnet of N_u, and

- $Min(N)$ is a set of conditions;
- every condition of N has at most one output event in N;
- every node of N_u that does not belong N either precedes or is in conflict with some node of N;
- p is the restriction of p_u to the nodes of N.

These conditions imply in particular that $Min(N)$ is a cut of N_u. Sometimes, we denote the minimal elements of N by $Min(\pi)$.

As in the interleaving case, a run represents one of the possible futures of the system from a certain reachable state.

6.1 The partial order interpretation

We interpret the logic $\forall L(O, F, U)$ on the set of runs of the unfolding β_u.

In correspondence with the interleaving interpretation we write for two runs π an π' having E and E' as sets of nodes, respectively,

- $\pi \sqsubseteq \pi'$ if $E' \subseteq E$ i.e. π' is a suffix of π and
- $\pi \xrightarrow{a} \pi'$ if $E \setminus E' = \{e\}$ and $l(e) = a$.

Note, that we could have formulated a more general notion of an execution step: $\pi \xrightarrow{A} \pi'$ where A is a multiset of actions and the set of nodes underlying A is a *co*-set in π. We refrained from considering this more concurrent version of a step to keep the same logic for the interleaving and the noninterleaving interpretation.

The denotation of a formula is a set of runs of β_u, defined according to exactly the same rules as in the interleaving case, but taking partial order runs instead of interleaving runs.

Let $\mathcal{E}$ be a BPP with leading variable $X = E$, and let $\mathcal{R}$ be the set of runs of its unfolding. We say that $\mathcal{E}$ satisfies a formula ϕ if

$$\forall \pi \in \mathcal{R} \,.\, Min(\pi) = Min(N_u) \Rightarrow \pi \in \|\phi\|$$

The runs π such that $Min(\pi) = Min(N_u)$ are, loosely speaking, those starting at the initial state.

In the sequel we refer to this definition as the *partial order interpretation* of $\forall L(O, F, U)$.

7 Decidability of the partial order interpretation

The key to prove the decidability of the partial order interpretation is to observe that the unfolding of a BBP is almost a bipartite labelled tree. We have that:

- the conditions of an unfolding have at most one input event, because unfoldings are occurrence nets;
- the events of the unfolding of a BPP have at most one input condition, because the transitions of the nets obtained from BPPs have one single input place.

The "almost" is due to the fact that an unfolding may have more than one minimal element. This is only a minor technical difficulty, which can be easily overcome by adding a 'junk' root node to the unfolding.

We now profit from the fact that the validity problem for the monadic second order logic of a tree with fan-out degree n, denoted by SnS, is decidable [19]. We shall reduce the model checking problem for the partial order interpretation of $\forall L(O, F, U)$ to this problem.

We first fix some notations on SnS. The language of SnS contains a constant ϵ, unary function symbols $succ_1, \ldots, succ_n$, a binary predicate symbol $\leq$ and an arbitrary finite set of unary predicate symbols. SnS is the monadic second order logic over this language; i.e. formulas are built from the symbols of the language, first-order variables $x, y, \ldots$, second order variables $X, Y, \ldots$ and the quantifiers $\exists$, $\forall$ (ranging over either kind of variable). Unary predicates can be interpreted as sets; according with it, we write $x \in P$ instead of $P(x)$.

The standard interpretation has $\{1, 2, \ldots, n\}^*$ as domain; ϵ is mapped to the empty string; for $i = 1, \ldots, n$, $succ_i$ is mapped to the function $\mathbf{succ}_i(x) = xi$; $\leq$ is mapped to the prefix relation on $\{1, 2, \ldots, n\}^*$. This structure is also known as the infinite tree of fan-out degree n.

We proceed as follows. Given a BPP $\mathcal{E}$ and a formula ϕ of $\forall L(O, F, U)$, we construct two formulae of SnS, where n is large enough, for instance the length of the description of $PN(\mathcal{E})$ (with numbers represented in unary). The first of these two formulae, which we call *Unf*, has a unique model, which is (isomorphic to) the unfolding β_u of $\mathcal{E}$. The second formula, which we call G_ϕ, has as models the unfoldings which satisfy ϕ. Once these two formulae have been constructed, the model checking problem reduces to showing that the formula $Unf(\mathcal{E}) \Rightarrow G_\phi$ is valid.

The definition of $Unf(\mathcal{E})$ is easy. Let $\{s_1, \ldots, s_k\}$ and $\{t_1, \ldots, t_l\}$ be the sets of places and transitions of $PN(\mathcal{E})$. We introduce for every place s_i a predicate P_{s_i}, for every transition t_j a predicate $P_{(t_j, l_{\mathcal{E}}(t_j))}$, and finally a predicate P_{junk} to identify junk nodes. We can easily express the following conditions in SnS:

- every node of the tree satisfes exactly one predicate,
- the root satisfies P_{junk},
- for every place s_i, the successors $M_0^{\mathcal{E}}(s_1) + \ldots + M_0^{\mathcal{E}}(s_{i-1})$ to $M_0^{\mathcal{E}}(s_1) + \ldots + M_0^{\mathcal{E}}(s_i)$ of the root satisfy P_{s_i}, and the rest of the successors satisfy P_{junk},
- for every place s_i, if a node satisfies P_{s_i}, then its successors $W_{\mathcal{E}}(s_i, t_1) + \ldots + W_{\mathcal{E}}(s_i, t_{j-1})$ to $W_{\mathcal{E}}(s_i, t_1) + \ldots + W_{\mathcal{E}}(s_i, t_j)$ satisfy $P_{(t_i, l(t_j))}$, and the rest of its successors satisfy P_{junk},
- for every transition t_j, if a node satisfies $P_{(t, l_{\mathcal{E}}(t_j))}$, then its successors $W_{\mathcal{E}}(t, s_1) + \ldots + W_{\mathcal{E}}(t, s_{i-1})$ to $W_{\mathcal{E}}(t, s_1) + \ldots + W(t, s_i)$ satisfy P_{s_i}, and the rest of its successors satisfy P_{junk},
- if a node different from the root satisfies P_{junk}, then its succesors satisfy P_{junk}.

$Unf(\mathcal{E})$ is the conjunction of these conditions. It is routine to see that its only model is the maximal branching process of $PN(\mathcal{E})$, once the junk nodes are removed.

We now introduce some auxiliary formulas of SnS. They contain some free variables; the name of the formula is parameterized with them.

The irreflexive prefix relation $<$ on $\{1,\ldots,n\}^*$ is definable in SnS. Using this fact, we can easily express that two nodes x, y are in conflict by the following formula $Conf(x,y)$:

$$\begin{array}{l} \exists z \,.\, \bigvee_{s \in S} \; z \in P_s \wedge \\ \qquad z < x \wedge z < y \;\wedge\; \neg(x < y) \vee \neg(y < x)\,) \end{array}$$

We now construct a formula $Run(X)$ which expresses that X is the set of nodes of a run. It suffices to require four conditions: X is conflict-free, its minimal elements are conditions, every element which does not belong to X is either smaller than or in conflict with some element of X and, finally, that X is upwards closed. In order to express the second condition, we construct the formula $Min(x, X)$, which expresses that x is a minimal element of X:

$$\forall y \,.\, y \in X \rightarrow \neg(y < x)$$

We define $Run(X)$ as the conjunction of the following formulae:

$$\begin{array}{l} \forall x \,.\, Min(x, X) \rightarrow \bigvee_{s \in S} x \in P_s \\ \forall x \, \forall y \,.\, (x \in X \wedge y \in X) \rightarrow \neg\, Conf(x,y) \\ \forall x \, \forall y \, \forall z \,.\, (x \in X \wedge y \in X \wedge x < z < y) \rightarrow z \in X \\ \forall x \,.\, \neg(x \in X) \rightarrow \exists y \,.\, y \in X \wedge (x < y \vee Conf(x,y)) \end{array}$$

Now we define the formula $Succ(X, Y, x)$.

$$\begin{array}{l} Y \subset X \;\wedge\; \forall y \,.\; (y \in X \wedge \neg(y \in Y) \wedge \\ \qquad \bigvee_{t \in T_{\mathcal{E}}} \; y \in P_{(t, l_{\mathcal{E}}(t))}) \rightarrow y = x \end{array}$$

With the help of these formulae, we encode the partial order interpretation of $\forall L(O, F, U)$ into SnS. To simplify the formulae, we assume that $\forall, \exists$ quantify over runs.

$$\begin{array}{rcl} F_{\mathbf{true}}(X) & = & Run(X) \\ F_{\neg\phi}(X) & = & \neg F_{\phi}(X) \\ F_{\phi_1 \wedge \phi_2}(X) & = & F_{\phi_1}(X) \wedge F_{\phi_2}(X) \\ F_{\forall\phi}(X) & = & \forall Y.(\forall x. Min(x, X) \leftrightarrow Min(x, Y)) \\ & & \rightarrow F_{\phi}(Y) \\ F_{(a)\phi}(X) & = & \exists Y. \exists x. Succ(X, Y, x) \\ & & \wedge \bigvee_{t \in l_{\mathcal{E}}^{-1}(a)} x \in P_{(t,a)} \;\wedge\; F_{\phi}(Y) \\ F_{\phi_1 \mathbf{U} \phi_2}(X) & = & \exists Y. X \subseteq Y \wedge F_{\phi_2}(Y) \wedge \forall Z. X \subseteq Z \\ & & \wedge Z \subseteq Y \rightarrow F_{\phi_1}(Z) \end{array}$$

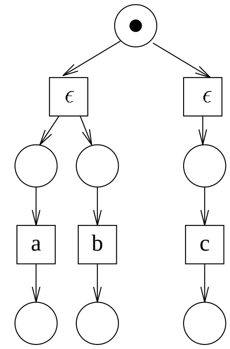

Figure 3: Net semantics of $(a \parallel b) + c$

Finally, since $\mathcal{E}$ satisfies a formula ϕ of $\forall L(O, F, U)$ if all the runs that start at the initial state are in $||\phi||$, we introduce a formula $IRun(X)$:

$$\begin{array}{l} Run(X) \;\wedge\; (\exists x \exists y \,.\, y < x \wedge Min(x, X)) \rightarrow \\ \qquad \forall z \,.\, \neg(z < y) \end{array}$$

i.e. $IRun(X)$ holds if X is a run and the only node of the tree smaller than some minimal element of X is the root.

We obtain:

Theorem 7.1 *Let $\mathcal{E}$ be a simple BPP and let ϕ be a formula of $\forall L(O, F, U)$. Then $\mathcal{E}$ satisfies ϕ iff the following formula of SnS is a tautology:*

$$Unf(\mathcal{E}) \rightarrow (\forall X \,.\, IRun(X) \rightarrow F_{\phi}(X))$$

There are no serious conceptual problems to extend this result to all BPPs. We can take the net semantics of CCS without restriction and relabelling given by Gorrieri and Montanari in [12], which associates to every BPP a finite Petri net. This semantics is more difficult to describe succintly than the one shown here, and that is why we have not considered it in the first place. It introduces some extra transitions that do not correspond to the execution of process actions. For instance, the non-simple BPP expression $(a \parallel b) + c$ is translated into the net of Figure 3. The unfoldings of the nets obtained with this semantics are again trees. It is easy (but tedious) to change the definition of the partial order interpretation to take into accout that an action corresponds to the atomic occurrence of several transitions.

A natural question to ask is why this decidability proof does not work in the interleaving case. The proof consists of three parts:

- BPPs are given a semantics with a tree structure,
- the tree is encoded into SnS, and
- the logic is encoded into SnS.

When we try to extend this decidability proof to the interleaving case, there are two possibilities. In the first one, we take the unfolding of the Petri net as semantics. As we have seen, this unfolding can be encoded into SnS. However, the interleaving interpretation of the logic cannot: it is not possible to replace $Run(X)$ by a formula $FiringSequence(X)$, because a firing sequence is not characterised by its set of events. In the second possibility, we take the unfolding of the transition system as semantics. Now, we can construct an SnS formula $FiringSequence(X)$, which holds for a set of reachable states X iff they are the states of a maximal path, but it is no longer possible to encode the unfolding as an SnS formula!

8 Conclusions

We have proved the undecidability of the model checking problem for the fragment $B(O, F)$ of the logic $\forall L(O, F, U)$ and VBPPs (recursive processes built out of atomic actions and the prefix and parallel operators) in the usual interleaving semantics. $B(O, F)$ corresponds to the fragment of CTL containing the operators AX and AF. This result shows that most branching time logics described in the literature become undecidable even for very simple infinite-state concurrent systems. The situation of the finite state case, in which branching time logics are easier to check than linear time ones, gets inverted, because the linear time mu-calculus, a rather powerful linear time logic, is decidable for BPPs, and even for Petri nets, which have larger expressive power [10].

We also show that $\forall L(O, F, U)$ is decidable for simple BPPs in a natural partial order semantics. The result follows easily from the fact that this semantics is always a tree expressible in SnS, the monadic second order logic of n successors.

This result is not as conclusive as the first, because BPPs have a limited expressive power, and we do not know how far can the decidability result be extended to larger classes of processes. However, it adds a new motivation for the study of partial order logics. So far, these logics have been studied either because they can express some properties difficult to formalise with interleaving logics like serializability of transactions, or concurrency of program segments [17, 18], or because they extend well-known interleaving logics [21]. In the finite state case, partial order logics tend to have higher complexity than interleaving logics. Our results show that in the infinite state case partial order logics may be easier to handle.

References

[1] J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. *Information and Computation* 60:109–137, 1984.

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

[3] S. Christensen, Y. Hirshfeld, and F. Moller. Bisimulation Equivalence is Decidable for all Basic Parallel Processes. In *Proceedings of CONCUR '93*, LNCS 715:143–157, 1993.

[4] S. Christensen, H. Hüttel, and C. Stirling. Bisimulation Equivalence is Decidable for all Context-free Processes. In *Proceedings of CONCUR '92*, LNCS 630:138–147, 1992.

[5] E.M. Clarke and E.A. Emerson. Design and Synthesis of synchronization skeletons using Branching Time Temporal Logic. In *Proceedings of Workshop on Logics of Programs*, LNCS 131:52–71, 1981.

[6] E.A. Emerson. Temporal and Modal Logic. In *Handbook of Theoretical Computer Science, Volume B*, 995–1072, 1990.

[7] E.A. Emerson and J.Y. Halpern. "Sometimes" and "Not Never" revisited: on Branching versus

Linear Time Temporal Logic. *Journal of the ACM* 33(1):151–178, 1986.

[8] E.A. Emerson and C. S. Jutla. Tree Automata, Mu-Calculus and Determinacy. In *Proceedings of FOCS '91*, 1991.

[9] J. Engelfriet. Branching processes of Petri nets. *Acta Informatica* 28:575–591, 1991.

[10] J. Esparza. On the Decidability of the Model Checking Problem for Several μ-calculi and Petri Nets. In *Proceedings of CAAP '94*, LNCS 787:115–129, 1994.

[11] J. Esparza. On the uniform word problem for commutative context-free grammars. Submitted for publication, 1994.

[12] R. Gorrieri and U. Montanari. A Simple Calculus of Nets. In *Proceedings of CONCUR '90*, LNCS 458:2–30, 1990.

[13] Y. Hirshfeld. Petri Nets and the Equivalence Problem. In *Proceedings of CSL '93*, 1994.

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

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

[16] M. Minsky: Computation. Finite and Infinite Machines. Prentice-Hall, 1967.

[17] D. Peled, S. Katz, and A. Pnueli. Specifying and Proving Serializability in Temporal Logic. In *Proceedings of LICS '91*, 232–245, 1991.

[18] W. Penczek. Temporal Logics for Trace Systems: On Automated Verification. *International Journal on Foundations of Computer Science* 33:31–67, 1992.

[19] M.O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society* 141:1–35, 1969.

[20] C. Stirling. Modal and Temporal Logics. In *Handbook of Logic in Computer Science*, Oxford University Press, 1991.

[21] P.S. Thiagarajan. A Trace Based Extension of PTL. In *Proceedings of LICS '94*, 1994.

[22] P. Wolper. Temporal Logic can be more expressive. *Information and Control* 56(1,2):72–93, 1983.