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 automata, 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

* This work was partially done within the Sonderforschungsbereich 342 WG A3: SAM

[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 left open.

In this contribution, we consider a logic $\forall L(O, F, U)$ 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, EX, and AF), is already undecidable 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 $\forall L(O, F, U)$ is decidable in the partial order interpretation (more precisely, we prove it for a subclass 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:

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

where a belongs to a set 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, it appears 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 $\mathcal{T} = (\mathcal{S}, \{\xrightarrow{a} \mid a \in Act\})$, whose states are the BPP expressions reachable from the leading variable, and whose transition relations are the least relations satisfying the following rules:

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

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

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 (O stands for the relativized next operator (a)) :

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

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

Let $\mathcal{T}$ be the transition system of a BPP $\mathcal{E}$ 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 $s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \ldots$ 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 $s_n \xrightarrow{a} s$. 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 $\pi \xrightarrow{a_0} \pi'$ if $\pi = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \ldots$ and $\pi' = s_1 \xrightarrow{a_1} \ldots$.

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

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

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 $\forall L(O, F, U)$:

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

Notice that this is a pure branching-time language, because the linear time operators (a) and F only appear quantified. We call it $B^-(O, F)$ (the extension of $B^-(O, F)$ containing also $\forall(a)\phi$ and $\exists F\phi$ is called $B(O, F)$ in [20]).

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 $\mathtt{M}$, and a formula *Halt* of $B^-(O, F)$ such that $\mathcal{M}$ halts if and only if the VBPP satisfies *Halt*.

In the sequel we identify this VBPP and its leading variable.

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

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

which expresses that $\mathtt{M}$ eventually reaches a state from which it can do $\mathtt{halt}$.

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 $\mathtt{M}$, 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 $\mathtt{M}$ whose states satisfy ϕ_h, and
(2) if all the states of a run starting at $\mathtt{M}$ satisfy ϕ_h, then the run is honest.

Then, we shall define

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

If the model $\mathtt{M}$ of the counter machine satisfies *Halt*, then the runs starting at $\mathtt{M}$ 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 $\mathtt{M}$ 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 $\mathtt{C_j}$.

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

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 rules.

$$\begin{array}{lll} \text{type } I: & \mathtt{SQ_i} \stackrel{\text{def}}{=} \mathtt{in_i} \cdot (\mathtt{SQ_i} \parallel \mathtt{Q_i}) & \mathtt{Q_i} \stackrel{\text{def}}{=} \mathtt{out_i} \cdot (\mathtt{Q_k} \parallel \mathtt{C_j}) \\ \text{type } II: & \mathtt{SQ_i} \stackrel{\text{def}}{=} \mathtt{in_i} \cdot (\mathtt{SQ_i} \parallel \mathtt{Q_i}) & \mathtt{Q_i} \stackrel{\text{def}}{=} \mathtt{out_i} \cdot \mathtt{0} \\ & \mathtt{SQ_{n+1}} \stackrel{\text{def}}{=} \mathtt{in_{n+1}} \cdot (\mathtt{SQ_{n+1}} \parallel \mathtt{Q_{n+1}}) & \mathtt{Q_{n+1}} \stackrel{\text{def}}{=} \mathtt{halt} \cdot \mathtt{0} \end{array}$$

VBPPs cannot model the fact that from a state q_i of type II either the state q_k or the state q'_k can be reached, because in order to describe the choice between q_k and q'_k the choice operator is needed.

The model $\mathtt{M}$ of the counter machine is defined by

$$\mathtt{SM} \stackrel{\text{def}}{=} \mathtt{SQ_1} \parallel \ldots \parallel \mathtt{SQ_{n+1}}$$
$$\mathtt{M} \stackrel{\text{def}}{=} \mathtt{SM} \parallel \mathtt{Q_0}$$

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 corresponds to the state, and the indices $j_1, \ldots, j_m$ correspond to 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

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

A state q_i of type II is modelled by

$$\mathtt{SQ_i} \stackrel{\text{def}}{=} \mathtt{in_i^1} \cdot (\mathtt{Q_i} \parallel \mathtt{SQ_i}) \quad \mathtt{Q_i} \stackrel{\text{def}}{=} \mathtt{out_i^1} \cdot \mathtt{out_i^2} \cdot \mathtt{0}$$

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

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 $\mathtt{out}_{\mathtt{i}}^{1}$, $\mathtt{out}_{\mathtt{i}}^{2}$, $\mathtt{dec}_{\mathtt{i}}^{2}$ or $\mathtt{dec}_{\mathtt{i}}^{3}$, and let $a_1, \ldots, a_k$ be actions of A. Define

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

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

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

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

and the second is

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

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

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

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

Proof. (Sketch). Consider an arbitrary meaningful state E of a run in which every state satisfies ϕ_h. 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, examine the actions enabled at E, and check that only one leads to a state E' satisfying ϕ_h. Then 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.

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 2. *The model checking problem for the logic* $B^{-}(O, F)$ *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 so called 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:

$$\begin{array}{rcl} S & ::= & \mathbf{0} \mid X \mid a \cdot E \mid S + S \\ E & ::= & S \mid E \parallel E \end{array}$$

In general, simple BPP processes are not finite-state but they can be characterized 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:

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

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 occurrences of an element G in the multiset $|E|$ is denoted by $|E|_G$.

Proposition 3. *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 $G \stackrel{a}{\rightarrow} H$ 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 $G \stackrel{a}{\rightarrow} H$ there is exactly one representation for H according to 3. (up to $\equiv$).*

5.1 The Partial Order Interpretation

For lack of space we refer to [9] for basic definitions and notations of net theory. We denote a *labelled net* by the fourtuple (S, T, W, l), where S, T are disjoint sets of *places* and *transitions*, $W: (S \times T) \cup (T \times S) \rightarrow \mathbb{N}$ is a *weight* function, and $l: T \rightarrow \mathcal{L}$ is a *labelling* function. A *Petri net* is a pair (N, M_0), where N is a net and M_0 is 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

$G \xrightarrow{a} H$ for a generator G, then the net contains a transition with G as input place. The set of the output places is obtained from the process H as follows. Due to Proposition 3 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. So the Petri net associated to $\mathcal{E}$ is $PN(\mathcal{E}) = (N(\mathcal{E}), M_0^{\mathcal{E}})$ where

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

and 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.

For example the net representation of

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

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 4. *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.*

The partial order counterpart of a labelled transition system is the *unfolding* of the Petri net [9]. The unfolding of a Petri net is an acyclic net, usually infinite. Figure 1 shows on the left a Petri net, and on the right an initial part of its infinite unfolding. We denote the unfolding of a net $PN(\mathcal{E})$ by $\beta_u = (N_u, p_u)$ where $N_u = (B, E, W)$ is the acyclic net structure and p_u the mapping establishing the link to the nodes of the net $PN(\mathcal{E})$. Following [9], the places and transitions of the unfolding of a Petri net are called *conditions* and *events*, respectively.

The *cuts* of the unfolding are the partial order counterparts of the states of the transition system. A maximal set $B' \subseteq B$ of conditions of N_u is a cut if

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

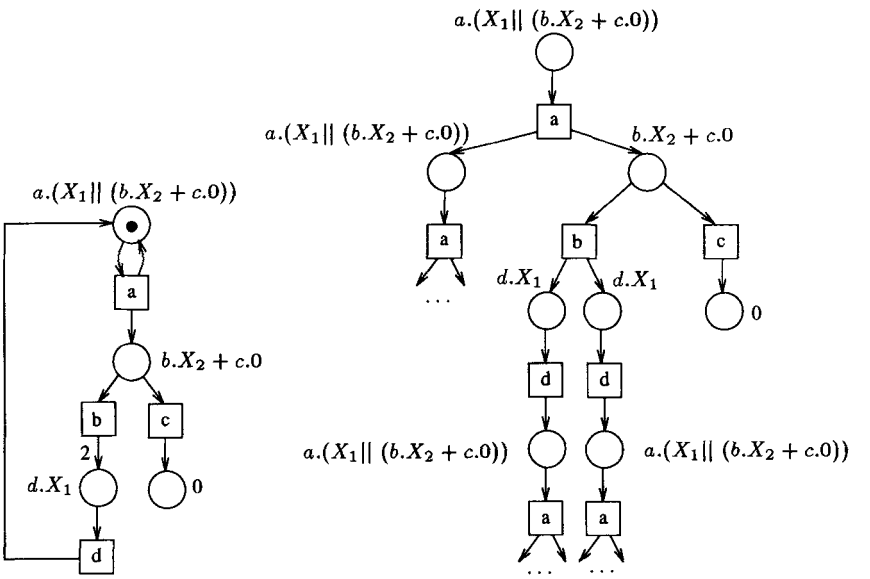

Fig. 1. Net representation of the BPP given in the text (left) and its unfolding (right)

where $x \preceq y$ if there is a path in N_u leading from x to y and $x \# y$ if there are $e, e' \in E$, $e \neq e'$ such that ${}^\bullet e = {}^\bullet e'$ and $e \preceq x$ and $e' \preceq y$ (we then say that x and y are in *conflict*).

There is a natural partial order on cuts which correponds to the reachability relation between states:

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

Finally, a *run* in the partial order interpretation of $\forall L(O, F, U)$ is a pair $\pi = (N, p)$, where N is a conflict free subnet of N_u satisfying

- $Min(N)$ is a cut of N_u, and
- N is not properly included in any other conflict free subnet N' such that $Min(N) = Min(N')$.

and p is the restriction of p_u to the nodes of N. $Min(N)$ is also denoted by $Min(\pi)$.

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

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 and 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 $p_u(e) = a$.

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. For a BPP $\mathcal{E}$ 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 $\forall L(O, F, U)$.

6 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 simple BBP is almost a bipartite labelled tree. The conditions of an unfolding have at most one input event, because unfoldings are occurrence nets; moreover the events of the unfolding of a simple 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 to 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.

For the definition of $Unf(\mathcal{E})$ we introduce for every place s_i of $PN(\mathcal{E})$ 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. It is now routine to construct a formula $Unf(\mathcal{E})$ such 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 free variables; the name of the formula is parameterized with them.

The irreflexive prefix relation $<$ on $\{1,\ldots,n\}^\star$ is definable in SnS. Using this fact, we can easily define formulas $Conf(x,y)$, $Min(x,X)$ and $Succ(X,Y,x)$ expressing that x and y are in conflict, x is minimal in X and that there is exactly one element $x \in \bigcup P_{(t,l_\mathcal{E}(t))}$ with $x \in X \setminus Y$ where $Y \subseteq X$, respectively. With the help of these we can define the formula $Run(X)$ as the conjunction of

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

and 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.

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

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 define a formula $IRun(X)$ for initial runs (those runs of the unfolding in which the minimal conditions have no input events).

Theorem 5. *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. The net semantics of CCS without restriction and relabelling given by Gorrieri and Montanari in [12] associates to every BPP a finite Petri net in which every transition has exactly one input place. This semantics is more difficult to describe succintly, and that is why we have not considered it here.

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!

7 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 EX 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.

Acknowledgements

We thank Colin Stirling and three anonymous referees for helpful comments.

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.