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

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

Abstract

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

1 Introduction

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

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

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

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

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

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

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

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

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

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

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

2 Models: notations and basic definitions

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

2.1 Transition systems

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

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

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

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

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

2.2 Petri Nets

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

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

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

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

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

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

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

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

2.3 Basic and Very Basic Parallel Processes

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

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

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

A BPP is a family of recursive equations

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

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

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

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

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

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

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

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

Also, it follows easily from this translation that:

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

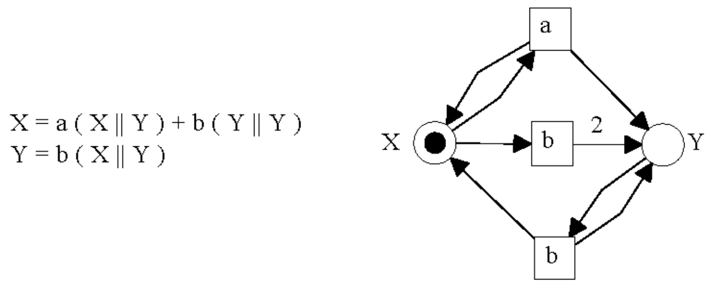

Fig. 1 A BPP and its corresponding Petri net

3 Linear Time Logics

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

The linear time mu-calculus has the following syntax

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

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

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

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

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

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 $||\phi||$.

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

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS

Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS

Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS

Sync your system to PACER to automate legal marketing.