More Infinite Results

Javier Esparza*
Institut für Informatik
Technische Universität München

Arcisstr. 21
D-80290 München, GERMANY

Abstract

Recently there has been a spurt of activity in concurrency theory centered on the analysis of infinite-state systems. The following two problems have been intensely investigated: (1) given two infinite-state systems, are they equal with respect to a certain equivalence notion?, and (2) given an infinite-state system and a property expressed in a certain temporal logic, does the system satisfy the property? In his CONCUR '96 paper, Faron Moller surveys some of the key results on the decidability and complexity of (1). The purpose of this paper for CONCUR's satellite INFINITY Workshop is to do the same with (2).

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 methods have been developed to overcome this limitation, at least for restricted classes of infinite-state systems. Using them, several verification problems have been shown to be decidable, and even tractable. These decidability and complexity results can be classified into two groups:

- results on the *equivalence problem*: given two infinite-state systems, are they equal with respect to a certain equivalence notion?
- results on the *model-checking* problem: given an infinite-state system and a property expressed in a certain temporal logic, does the system satisfy the property?

*Partially supported by Teilprojekt A3 SAM of the Sonderforschungsbereich 342.

In his CONCUR'96 paper [23], Faron Moller surveys some of the most relevant results of the first group. The purpose of this paper is to give a similar overview of the results of the second group.

The model-checking problem has two parameters: a family of systems, and a temporal logic used to express properties of their behaviour. Following Faron Moller [23] (who takes this idea from Didier Caucal), we consider families of transition systems defined by rewrite rules including sequential rewrite transition systems, like pushdown processes and BPA-processes, and parallel rewrite transition systems, like Petri nets and Basic Parallel Processes. Temporal logics are classified according to their linear-time or branching-time character, and to their expressive power.

2 Rewrite Transition Systems

The contents of this section are taken almost verbatim from [23]. The definitions of labelled transition system and labelled rewrite transition system have been slightly simplified.

Definition 2.1 A *labelled transition system* is a tuple $\langle S, \Sigma, \longrightarrow, \alpha_0 \rangle$ where

- S is a set of *states*.
- Σ is a finite set of *labels* or *actions*.
- $\longrightarrow \subseteq S \times \Sigma \times S$ is a *transition relation*, written $\alpha \stackrel{a}{\longrightarrow} \beta$ for $\langle \alpha, a, \beta \rangle \in \longrightarrow$.
- $\alpha_0 \in S$ is a distinguished *start state*.

Definition 2.2 A *sequential labelled rewrite transition system* is a tuple $\langle V, \Sigma, P, \alpha_0 \rangle$ where

- V is a finite set of *variables*; the elements of V^* are referred to as *states*.
- Σ is a finite set of *labels* or *actions*.
- $P \subseteq V^* \times \Sigma \times V^*$ is a finite set of *rewrite rules*, written $\alpha \stackrel{a}{\longrightarrow} \beta$ for $\langle \alpha, a, \beta \rangle \in P$, which are extended by the *prefix rewriting rule*: if $\alpha \stackrel{a}{\longrightarrow} \beta$ then $\alpha\gamma \stackrel{a}{\longrightarrow} \beta\gamma$.
- $\alpha_0 \in V^*$ is a distinguished *start state*.

A *parallel labelled rewrite transition system* is defined precisely as above, except that the elements of V^* are read modulo commutativity of catenation, which is thus interpreted as parallel, rather than sequential, composition.

The families of transition systems which can be defined by restricted rewrite systems can be classified using a form of Chomsky hierarchy. (Type 1—context-sensitive—rewrite systems do not feature in this hierarchy since the rewrite rules by definition are only applied to the prefix of a composition.) This hierarchy provides an elegant classification of

several important classes of transition systems which have been defined and studied independent of their appearance as particular rewrite systems. This classification is presented as follows.

	Restriction on the rules $\alpha \xrightarrow{a} \beta$ of P	Sequential composition	Parallel composition
Type 0:	*none*	PDA	PN
Type 2:	$\alpha \in V$	BPA	BPP
Type 3:	$\alpha \in V,\ \ \beta \in V \cup \{\varepsilon\}$	FSA	FSA

FSA represents the class of finite-state automata. Clearly if the rules are restricted to be of the form $A \xrightarrow{a} B$ or $A \xrightarrow{a} \varepsilon$ with $A, B \in V$, then the reachable states of both the sequential and parallel transition systems will be a subset of the finite set of variables V. (We assume here that the initial state itself is a member of V.)

BPA represents the class of Basic Process Algebra processes of Bergstra and Klop [2], which are the transition systems associated with Greibach normal form (GNF) context-free grammars in which only left-most derivations are permitted. BPA-processes are also called context-free processes in the literature.

BPP represents the class of Basic Parallel Processes introduced by Christensen [15] as a parallel analogy to BPA, and are defined by the transition systems associated with GNF context-free grammars in which arbitrary grammar derivations are permitted.

PDA represents the class of push-down automata which accept on empty stack. To present such PDA as a restricted form of rewrite system, we first assume that the variable set V is partitioned into disjoint sets Q (finite control states) and Γ (stack symbols). The rewrite rules are then of the form $pA \xrightarrow{a} q\beta$ with $p, q \in Q$, $A \in \Gamma$ and $\beta \in \Gamma^*$, which represents the usual PDA transition which says that while in control state p with the symbol A at the top of the stack, you may read the input symbol a, move into control state q, and replace the stack element A with the sequence β. Caucal [12] demonstrates that any unrestricted (type 0) sequential rewrite system can be presented as a PDA, in the sense that the transition systems are isomorphic up to the labelling of states.

PN represents the class of (finite, labelled, weighted place/transition) Petri nets, as is evident by the following interpretation of unrestricted parallel rewrite systems. The variable set V represents the set of places of the Petri net, and each rewrite rule $\alpha \xrightarrow{a} \beta$ represents a Petri net transition labelled a with the input and output places represented by α and β respectively, with the weights on the input and output arcs given by the relevant multiplicities in α and β. Note that a BPP is a communication-free Petri net, one in which each transition has a unique input place.

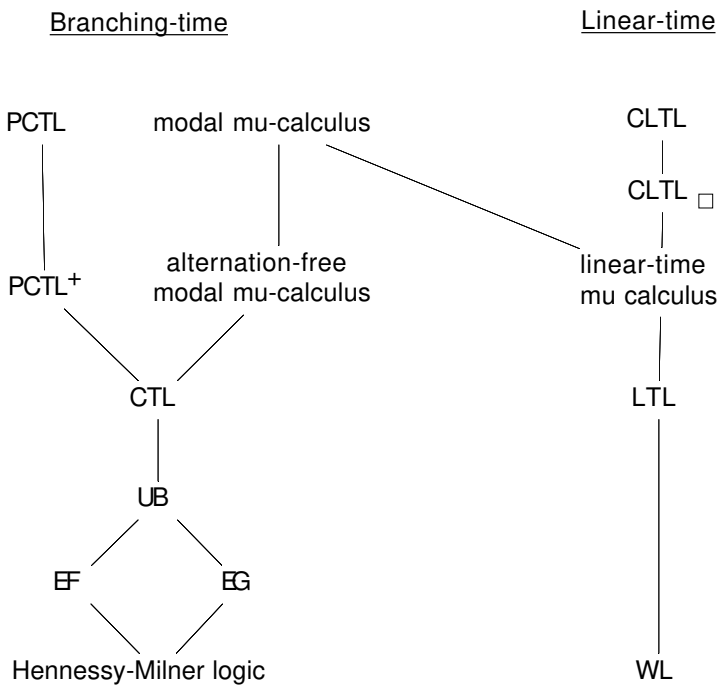

Figure 1: Linear and branching-time logics

3 Temporal logics

Temporal logics can be given a *state-based* or an *action-based* semantics, or a combination of the two. In state-based semantics, formulae are built out of a set of *atomic sentences* or *propositions*, and are interpreted according to a *valuation*, which assigns to each atomic sentence a set of states of the transition system. The information carried by the labels on top of the arrows is ignored (actually, it is assumed that the semantics of a system is an unlabelled transition system). In action-based semantics, the only atomic sentence is *true*, and so the information carried by the states is ignored. Logics using this semantics have *relativised* next operators, one for each possible label.

Action-based semantics is used in this paper, for two reasons. First, it seems to be more natural than state-based semantics for rewrite transition systems. Second, and more important, the decidability of the model-checking problem for a given logic may heavily depend on the set of atomic sentences. Therefore, in our analysis of the model-checking problem we would have to consider the set of atomic sentences as an additional parameter, which would complicate the presentation of the results.

Figure 1 shows a classification of the temporal logics we are going to introduce according to their linear-time or branching-time nature, and to their expressiveness. A line between two logics indicates that the one placed higher up is strictly more expressive. LTL and CTL are included because they are very popular, although strictly speaking they do not play a rôle in this paper.

The logics are introduced informally. Most formal definitions can be found, for instance,

in the contributions of E. Allan Emerson and Colin Stirling to [29]. When this is not the case, a reference is given.

Linear-time logics A *path* of a labelled transition system is a finite or infinite sequence $s_0 \stackrel{a_0}{\longrightarrow} s_1 \stackrel{a_1}{\longrightarrow} s_2 \ldots$ such that every triple $s_i \stackrel{a_i}{\longrightarrow} s_{i+1}$ belongs to the set of transitions. A *run* is a *maximal* path, i.e., a path which is either infinite, or terminates on a state without successors. A formula of a linear-time temporal logic is interpreted on the runs of a transition system. A transition system satisfies a formula if every run starting at the initial state satisfies it.

The weakest linear-time logic, which is called WL here, is built out of *true*, boolean operations (including negation), and a next operator $(a)\phi$ for each action a. A run satisfies $(a)\phi$ if its first action is a, and the suffix run obtained after chopping off the first state and the first action satisfies ϕ.

Linear Temporal Logic (LTL) is obtained by adding to WL an *until* operator $\phi\, U\, \psi$. A run $s_0 \stackrel{a_0}{\longrightarrow} s_1 \stackrel{a_1}{\longrightarrow} \ldots\ldots$ satisfies $\phi\, U\, \psi$ if ϕ holds until ψ holds, i.e., if some suffix $s_i \stackrel{a_i}{\longrightarrow} s_{i+1} \stackrel{a_{i+1}}{\longrightarrow} \ldots$ satisfies ψ, and all suffixes $s_j \stackrel{a_j}{\longrightarrow} s_{j+1} \stackrel{a_{j+1}}{\longrightarrow} \ldots$ with $j < i$ satisfy ϕ. Derived operators are $F\phi = true\, U\, \phi$ (eventually ϕ) and its dual $G\phi = \neg F \neg \phi$ (always ϕ).

The *linear-time mu-calculus* is obtained by adding variables to WL, together with the greatest fixpoint operator $\nu Z.\phi$, which binds the variable Z. A formula is *well-formed* if every variable is within the scope of an even number of negations, and *closed* if every variable is bound by a fixpoint operator. When we speak of the model-checking problem for the linear-time mu-calculus, we mean the model-checking problem for well-formed, closed formulae.[1]

Constrained Linear Temporal Logic (CLTL) is introduced in [5]. It extends LTL with two kinds of constraints: *pattern constraints*, which use finite state Rabin-Scott automata to impose structural constraints on runs, and *counting constraints*, which use Presburger arithmetic formulas to express constraints on the number of occurrences of events. In the fragment $\mathrm{CLTL}_{\Box}$, counting constraints cannot be introduced in eventuality formulae. $\mathrm{CLTL}_{\Box}$, and therefore CLTL, is strictly more expressive than the linear-time mu-calculus. A typical property of runs which is expressible in $\mathrm{CLTL}_{\Box}$ but not in the linear-time mu-calculus is 'every finite prefix contains at least as many a actions as b actions'.

Branching-time logics A formula of a linear-time temporal logic is interpreted on the states of a transition system. Branching-time operators are obtained from linear-time operators by quantifying on the runs starting at a given state. For instance, from the operator $(a)\phi$ one gets the operator $E(a)\phi$ ($\mathbf{A}(a)\phi$), which holds at a state s if some run (all runs) starting at s satisfy $(a)\phi$. (The notation $E(a)\phi$ is a compromise between the state-oriented operator $EX\phi$ of logics like CTL and the operator $\langle a \rangle \phi$ of Hennesy-Milner logic.) Notice that $A(a)\phi = \neg E(a) \neg \phi$.

The weakest branching-time logic is *Hennesy-Milner logic*, built out of *true*, boolean operations (including negation), and the existential next operator $E(a)\phi$ introduced above.

[1] We also apply this convention to the modal mu-calculi introduced below.

The *Unified System of Branching-Time Logic* (UB) [1] adds to Hennesy-Milner logic the operators $EF\phi$ and $EG\phi$. The fragments of UB containing only the operator $EF\phi$ ($EG\phi$) is called EF (EG).

Computation Tree Logic (CTL) adds to Hennesy-Milner logic an *until* operator $E\phi U\,\psi$.

Presburger Computation Tree Logic (PCTL) [4] extends CTL with counting constraints on actions expressed using Presburger arithmetic. The fragment PCTL$^+$ is obtained by imposing the following syntactic restriction: in every subformulae of the form $E\phi U\,\psi$, the formula ψ is a counting constraint, which in particular contains no temporal operators.

The *modal mu-calculus* extends Hennesy-Milner logic with greatest fixpoints, exactly as in the case of the linear-time mu-calculus. Least fixpoints are the dual of greatest fixpoints: $\mu X.\phi = \neg\nu X.\neg\phi[\neg X/X]$, where $\phi[\neg X/X]$ is the result of substituting $\neg X$ for X in ϕ. A formula is in *positive normal form* if it contains no negations, and using least fixpoints every formula can be put in positive normal form.

A formula in positive normal form is *alternation-free* if no ν-subformula has a free variable which, in the context of the whole formula, is bound by a μ, and viceversa. The *alternation-free modal mu-calculus* is the set of all alternation-free formulae.

4 Results for Type 0 Rewrite Systems

4.1 Pushdown automata

Branching-time logics It is a folklore result that the model-checking problem for the modal mu-calculus and pushdown automata is decidable: in [31] it is shown that the the monadic second order theory of a certain class of graphs, including the transition systems generated by pushdown automata, is decidable; since the modal mu-calculus can be embeded into monadic second order logic, the decidability of the modal mu-calculus follows.

The algorithm derived from this decidability proof is extremely inefficient: it has non-primitive recursive complexity. Igo Walukiewicz has recently shown in [34] that the model-checking problem for the modal mu-calculus and pushdown processes is DEXPTIME-complete. Hardness is proved by reduction from the problem of deciding if an alternating linear space bounded Turing Machine accepts an input. Membership in DEXPTIME is proved in three steps:

- The model checking problem is reduced to the existence of a winning strategy for Player I in a certain *parity game*. The board of the game is the transition system of the pushdown automaton, together with a priority function which assigns natural numbers to the states of the automaton. A move consists of selecting a state of the transition system reachable from the current one. Players I and II alternate their moves. A player that cannot do any move loses. If the game does not terminate, the winner is decided but looking at the smallest priority that appears infinitely often in the infinite path generated by the game. Player I wins if this priority is even, and Player II if it is odd.

- The existence of a winning strategy for Player I is reduced to the existence of a winning strategy in a finite game (a game with finite board).
- The existence of a winning strategy in the finite game is reduced to the model-checking problem of the modal mu-calculus over finite transition systems.

It is also shown in [34] that the model-checking problem remains DEXPTIME-hard for the alternation-free mu-calculus; even more, the problem needs exponential time in the size of the system.

The model-checking problem for the alternation-free fragment has also been studied by Olaf Burkhard and Bernhard Steffen in [11], where they present a global model-checking algorithm. The algorithm is an extension of a similar algorithm for BPA-processes, which is discussed in the next section.

Ahmed Bouajjani and Oded Maler present at INFINITY'96 a very elegant model-checking algorithm for pushdown processes and CTL [7]. They show that the set of configurations satisfying a property can be recognized by essentially an alternating finite automaton, and show how to compute the automaton corresponding to a formula from the automata corresponding to its subformulae.

Linear-time logics The decidability of the linear-time mu-calculus for pushdown processes follows from the decidability of the modal mu-calculus, but can also be easily established using the automata-theoretic approach to model-checking. Given a formula ϕ of the linear-time mu-calculus, we build a Büchi automaton $A_{\neg\phi}$ which accepts all computations that do not satisfy ϕ. Then, we construct the product of $A_{\neg\phi}$ and the pushdown process to yield an ω-pushdown automaton A_P. The formula ϕ holds if and only if A_P is empty, a decidable problem. This algorithm shows that the model-checking problem can be solved in PSPACE. Together with the PSPACE-hardness bound of [33] for LTL and finite-state systems, we obtain that the model-checking problem is PSPACE-complete.

In [6], Ahmed Bouajjani and Peter Habermehl define the fragment $\mathrm{CLTL}_{\Box}$ of CLTL, in which counting constraints cannot be introduced in eventuality formulae. They prove that $\mathrm{CLTL}_{\Box}$ is decidable for pushdown processes (CLTL is undecidable even for finite-state systems). The technique consists of showing that any formula of $\mathrm{CLTL}_{\Box}$ can be transformed into a conjunction of two formulae, the first of which is equivalent to a formula of the linear-time mu-calculus, while the second is a pure counting constraint. Since the linear-time mu-calculus is decidable, it remains to deal with the counting constraint. For that, they show that the model-checking problem for counting constraints is decidable for *semilinear systems*, which are those that generate sets of finite traces whose Parikh images are semilinear and effectively constructible. Finally, they use Parikh's classical theorem, which states that pushdown automata are semilinear.

Ahmed Bouajjani and Oded Maler show in [7] that the model-checking technique for CTL described above can also be applied to model-checking the linear-time mu-calculus, or in general any linear-time temporal logic which can be encoded into Büchi automata.

4.2 Petri Nets

While all the logics of Figure 1 below and including the modal mu-calculus were decidable for pushdown processes, the situation changes rather drastically when we move to Petri nets. The linear-time temporal logics remain decidable, but all the branching time logics (except, of course, Hennesy-Milner logic) become undecidable. A survey of these results has been given by the author in [17], and most of what follows is taken from there.

Branching-time logics The undecidability of nearly all the branching-time logics in Figure 1 is shown by reduction from the halting problem for Minsky machines [28] (also called counter or register machines in the literature). This is also the technique used to prove the undecidability of equivalence notions for Petri nets (see Moller's survey [23]). The following description of Minsky machines is taken verbatim from [23].

Minsky Machines [28] are simple straight-line programs which make use of only two counters. Formally, a *Minsky machine* is a sequence of labelled instructions

$$\begin{array}{ll} \mathtt{X}_0 & : \mathrm{comm}_0 \\ \mathtt{X}_1 & : \mathrm{comm}_1 \\ & \cdots \\ \mathtt{X}_{n-1} & : \mathrm{comm}_{n-1} \\ \mathtt{X}_n & : \mathtt{halt} \end{array}$$

where each of the first n instructions is either of the form

$$\mathtt{X}_\ell \ : \ \mathtt{c}_0 := \mathtt{c}_0{+}1; \ \mathtt{goto}\ \mathtt{X}_j \qquad \text{or} \qquad \mathtt{X}_\ell \ : \ \mathtt{c}_1 := \mathtt{c}_1{+}1; \ \mathtt{goto}\ \mathtt{X}_j$$

or of the form

$$\begin{array}{ll} \mathtt{X}_\ell \ : \ \mathtt{if}\ \mathtt{c}_0 = 0\ \mathtt{then}\ \mathtt{goto}\ \mathtt{X}_j & \text{or} \quad \mathtt{X}_\ell \ : \ \mathtt{if}\ \mathtt{c}_1 = 0\ \mathtt{then}\ \mathtt{goto}\ \mathtt{X}_j \\ \quad \mathtt{else}\ \mathtt{c}_0 := \mathtt{c}_0{-}1; \ \mathtt{goto}\ \mathtt{X}_k & \qquad\qquad\quad \mathtt{else}\ \mathtt{c}_1 := \mathtt{c}_1{-}1; \ \mathtt{goto}\ \mathtt{X}_k \end{array}$$

A Minsky machine M starts executing with the value 0 in the counters $\mathtt{c}_0$ and $\mathtt{c}_1$ and the control at the label $\mathtt{X}_0$. When the control is at label $\mathtt{X}_\ell$ ($0 \leq \ell < n$), the machine executes instruction comm_ℓ, modifying the contents of the counters and transferring the control to the appropriate label as directed by the instruction. The machine halts if and when the control reaches the `halt` instruction at label $\mathtt{X}_n$.

Petri nets cannot faithfully simulate Minsky machines, because Minsky machines are Turing powerful and Petri nets are not [32]. However, given a Minsky machine, it is easy to build a Petri net that 'almost' simulates it. In fact, it faithfully simulates all features of a Minsky Machine but one: the ability to test if the value of a counter is zero. More precisely, in an instruction of the second form, the simulating Petri net is free to branch to X_j even if the value of the counter is *not* 0. Therefore, the Petri net has 'honest' runs, in which the net branches to X_j only when the value of the corresponding counter is 0, and 'cheating' runs in which this rule is not respected. If a temporal logic is expressive enough to model the property 'honest runs terminate', then its model-checking problem for Petri nets is undecidable. Since termination can be easily expressed in all the branching-time

logics of Figure 1 above Hennesy-Milner logic, the key issue is whether the logic contains a formula which is satisfied by all and only the honest runs.

A simple inspection of the Petri net which simulates the Minsky machine shows that honest runs can be characterized as those satisfying a set of properties of the following form: at every state s reached along the run, if an action a representing the branch to X_j is enabled, then an action b, which can occur if and when the counter is nonzero, is disabled. These properties can be easily encoded using the operators $EG\phi$, $E(a)\phi$ and $E(b)\phi$, and therefore the model-checking problem for any logic into which these operators can be expressed is undecidable.

This argument shows undecidability of all the branching-time logics of Figure 1, with the exception of EF. Undecidability is proved in this case by reduction from the *marking equivalence* problem: given two Petri nets with the same set of places, do they have the same sets of reachable markings? (The undecidability of the marking equivalence problem was shown by Rabin [22], by means of a not so easy reduction from Hilbert's tenth problem.)

Linear-time logics Since the only atomic sentence of our logics is *true*, linear-time logics cannot express 'at the current state, if a is enabled then b is disabled': they can only express something about the next action that is executed in the run, but not about which were the possible alternatives to that action. Therefore, the argument used to prove the undecidability of branching-time logics does not apply anymore. In fact, it turns out that all the linear-time temporal logics of Figure 1 with the exception of CLTL are decidable.

The decidability of the linear-time mu-calculus, which implies the decidability of the other logics, was proved in [16] using the automata-theoretic approach. As before, given a formula ϕ of the linear-time mu-calculus, we build a Büchi automaton $A_{\neg\phi}$ which accepts all computations that do not satisfy ϕ. Then, we construct the product of $A_{\neg\phi}$ and the Petri net N being verified, to yield another Petri net N_P. An inspection of N_P shows that the net N satisfies ϕ if and only if at least one of a certain number of instances of the following two problems has a solution:

(1) given a Petri net and a place p, is there a reachable marking which marks p, and does not enable any transition?

(2) given a Petri net and a place p, is there an infinite computation such that infinitely many of the markings reached along it mark p?

Both (1) and (2) are known to be decidable. (2) can be solved in exponential space [35]; (1) can be reduced to the reachability problem, which is decidable [24] and requires at least exponential space [30], but for whose complexity no primitive recursive upper bound has been found so far. Unfortunately, (1) cannot be replaced by any much simpler problem because the model-checking problem for the linear-time mu-calculus is at least as hard as reachability.[2]

[2]This follows from the fact that the linear-time mu-calculus can express deadlock-freeness, and the

It is however possible to identify a fragment of the linear-time mu-calculus which can be reduced to (2) only. For this fragment, Richard Mayr has recently obtained a decidable tableau-system [26].

Ahmed Bouajjani and Peter Habermehl show in [6] that the model-checking problem is still decidable for the fragment $\mathrm{CLTL}_{\Box}$ of CLTL. Since Petri nets are *not* semilinear systems, they cannot apply the proof technique they used to prove the decidability of $\mathrm{CLTL}_{\Box}$ for pushdown processes. This time, the result is proved by means of a reduction to the reachability problem.

5 Results for Type 2 Rewrite Systems

In this section only branching-time logics are considered, for two reasons:

- pushdown automata are trace equivalent to BPA-processes, and therefore the results of Section 1 for linear-time logics apply to BPA-processes too; and,
- the model-checking problem for linear-time logics and BPPs has not been studied yet; the only known results are specialisations of decidability results for Petri nets.

5.1 Processes of Basic Process Algebra

Didier Caucal observed in [12] that pushdown automata are strictly more expressive than BPA-processes with respect to bisimulation equivalence. So there may exist simpler model-checking algorithms for BPA-processes than for pushdown processes. This question remains open for the modal mu-calculus, because the complexity of the model-checking problem for BPA-processes and the modal mu-calculus has not been established yet.

Olaf Burkart and Bernhard Steffen present in [10] an algorithm for BPA-processes and the alternation-free mu calculus which needs only *linear* time in the size of the system. This is a surprising result, since, as shown by Igo Walukiewicz, all algorithms for the same logic and pushdown processes need *exponential* time in the size of the system. So, moving down from pushdown processes to BPA-processes does indeed make model-checking easier in this case.

The algorithm of [10] works on formulations of BPA-processes and alternation-free formulas which are rather different from those used in this paper. Formulae are presented as *hierarchical equational blocks*, which we do not discuss further. Processes are encoded into a set of *procedural process graphs* (PPGs). A PPG is a transition system with an initial and a final state[3], whose transitions are labelled either with an action, as usual, or with the name of a PPG. The latter case models a procedure call. Intuitively, executing a transition $\sigma \xrightarrow{P} \sigma$, where P is the name of a PPG, means jumping from σ to the initial

reachability problem for Petri nets can be reduced in polynomial time to the deadlock-freeness problem [14].

[3]States of PPGs are called *state classes* in [10], and are denoted with the greek letter σ.

state of P, computing in P until termination (procedures may also not terminate), and then jumping back from the final state of P to σ'. Formally, the transition system of a PPG can be generated by iteratively replacing procedure calls by the called procedures.

The model-checking algorithm proceeds by iteratively computing a *property transformer* for each state of a PPG. A property transformer for a state σ takes a set of formulae M as arguments, and yields the set of formulas valid at σ under the assumption that all formulas of M are valid at the final state. Once the property transformers have been computed, the model-checking problem is solved by taking the property transformer corresponding of the initial state of the 'main procedure', and applying it to the set of formulae satisfied by the final state (which can be easily computed, because it is a deadlocked state).

This model-checking algorithm is *global*, because it provides complete information about the formulae satisfied by all the states of all the PPGs. In [21], Hardi Hungar and Bernhard Steffen present a local model-checking alternative to [10] in the form of a remarkably simple tableau system. The sequents of the tableaux are again inspired by the notion of property transformer. They are of the form $\sigma \vdash \langle \phi, \psi \rangle$, with intended meaning ϕ holds at state σ of the current PPG under the assumtion that ψ holds at the final state. The heart of the tableau system is the following rule, clearly inspired by the sequential composition rule of Hoare logic:

$$\frac{\sigma \vdash \langle \langle a \rangle \phi, \theta \rangle}{\sigma_P^s \vdash \langle \langle a \rangle \phi, \psi \rangle \quad \sigma' \vdash \langle \psi, \theta \rangle} \quad \sigma \xrightarrow{P} \sigma'$$

where σ_P^s denotes the initial state of the procedure P. The rule has the following intended meaning. In order to know if $\langle a \rangle \phi$ holds at σ assuming that θ holds at the final state, we guess an *intermediate assertion* ψ, and prove the following:

- if θ holds at the final state, then ψ holds after execution of the procedure P;
- if ψ holds after execution of the procedure P, then $\langle a \rangle$ holds immediately before its execution, i.e., $\langle a \rangle$ holds at σ.

Ahmed Bouajjani, Rachid Echahed and Riadh Robbana show in [4] that the logic PCTL is undecidable for BPA-processes by means of a rather straightforward reduction from the halting problem for Minsky machines. They also show that PCTL$^+$ is decidable by reduction to the validity problem of Presburger arithmetic.

5.2 Basic Parallel Processes

Branching-time logics In [18], Astrid Kiehn and the author show that the model-checking problem for the logic EG is also undecidable for BPPs, even for deterministic ones. This is a rather surprising result, because (deterministic) BPPs are a very weak model of computation. The result is again obtained by a reduction from the halting problem for Minsky machines. So, given a Minsky machine, a BPP is constructed, which simulates it. Since BPPs have much less modelling power than Petri nets, the simulating

BPP is not as faithful as the simulating Petri net. It can 'cheat' by jumping to X_j even when the corresponding counter is nonzero, but also by increasing a counter by more than 1, or by moving to a new state without performing any operation on the counter at all. Rather surprisingly, it is still possible to characterise the 'honest runs' in the logic EG, although the formula becomes very complicated.

After this result, the only branching-time logic of Figure 1 which remains to be explored is EF. It was shown in [17] that the model-checking problem for this logic is PSPACE-hard even for *finite* BPPs (BPP may encode finite-state systems much more succintly than finite automata). Richard Mayr has recently shown that the problem is PSPACE-complete [25], which implies that the addition of recursion to finite-state BPPs does not increase the complexity of model-checking EF. Mayr's proof combines two interesting results about BPPs:

- if $\{a_1, \ldots, a_k\}$ is the set of labels of a BPP N and $(n_1, \ldots, n_k)$ is a vector of natural numbers, it can be decided in polynomial time if $N \stackrel{w}{\longrightarrow}$ for some sequence of labels w containing n_i times each label a_i;
- if a BPP N of size n satisfies a formula $EF\phi$, then there is a sequence of length $O(2^{n^2})$ that leads to a configuration satisfying ϕ.

6 Systems out of the hierarchy

The model-checking problem has also been studied for some infinite systems out of the hierarchy of Section 2.

Arbitrary systems Julian Bradfield has proposed in [3] a sound and complete tableau system for the modal mu-calculus and arbitrary infinite transition systems. The tableau system is of course highly undecidable, but allows to structure arbitrary proofs.

PA systems PA (Process Algebra) is the name that has become common use to denote the algebra with a sequential and a parallel composition operator (without communication), plus recursion. The transition systems of this class of processes properly contains all those generated by Type 2 rewrite transition systems. Ahmed Bouajjani and Peter Habermehl observe in [6] that the model-checking problem for PA processes and LTL is undecidable. The result follows from the fact that halting of a Minsky machine can be reduced to the nonemptiness of the intersection of an ω-star-free language (which corresponds to some LTL formula) and a PA language.

Since PA properly includes BPP, the only logic of Figure 1 that might still be decidable for PA is the branching-time logic EF. Richard Mayr has very recently shown that this is indeed the case [27].

Extensions of BPA-processes Hardi Hungar has extended the tableau system of [21] for BPA-processes and the alternation-free modal mu-calculus in two different directions:

In [20], he introduces *higher-order processes*. Recall that BPA-processes were seen in [10] as procedural graphs. A transition of a procedural graph may be labelled with another graph. This label corresponds to a procedure call, and the called graph plays the rôle of the procedure body. In higher-order processes, the labels may be procedure calls with procedure parameters. For instance, a transition can call a procedure P with parameter Q, which causes that during the execution of P the transitions labelled with a procedure variable call the procedure Q.

In [19], he provides a sound and complete tableau system for parallel compositions of BPA-processes and the alternation-free modal mu-calculus. Notice that in this model BPA-processes *communicate* on common actions, which distinguishes it from BPPs, where there is no communication whatsoever between parallel processes. The tableau system cannot be decidable, because the parallel composition of two BPA-processes (with communication) can simulate a Turing Machine. It is shown to be decidable for the special case in which at most one of the communicating processes is infinite-state.

Infinite Graphs Transition systems are graphs, and graphs can be generated not only by means of structured operational semantics, or prefix rewriting rules.

Didier Caucal presents in [13] different classes of infinite graphs for which monadic second order logic is decidable; in particular, the paper introduces two powerful graph transformations which preserve the decidability of the monadic theory. The decidability of the monadic theory implies the decidability of the modal-mu calculus.

Olaf Burkart and Y.M. Quemener present at the INFINITY'96 a paper on model-checking graphs generated by graph grammars [9], which the author has not had time to read yet.

7 Conclusions

This paper has surveyed work on the model-checking problem for infinite-state systems. It is almost sure that, despite my efforts, I have forgotten some relevant work, cited incorrectly, and been unfair to the papers I am less familiar with. I apologise in advance for all that; I will be very grateful for corrections and suggestions.

I would like to condense the many results cited in the paper into three main conclusions. The first is:

> *All logics are decidable for sequential Type 0 systems, while only linear-time logics are decidable for parallel Type 0 systems.*

which is supported by the following three results:

- The modal mu-calculus is decidable for sequential Type 0 systems.

- (An extension of) The linear-time mu-calculus is decidable for parallel Type 0 transition systems.
- No branching-time logic[4] is decidable for parallel Type 0 systems.

The second conclusion is:

> *The complexity of the decidable logics is higher for parallel systems than for sequential systems.*

which is supported by the following two results:

- The model-checking for the modal mu-calculus and sequential systems is in DEXPTIME.
- The model-checking problem for all linear-time logics and Petri nets is at least EXPSPACE-hard.

The third (and for me rather surprising) conclusion is:

> *The decidability results change only very little when moving from Type 0 to Type 2 systems.*

which is supported by the result showing that EF is the only decidable branching-time logic for BPPs.

If we take into account that research on model-checking problems for infinite-state systems has a very short life, the collection of results of this paper is very impressive. But there still exist several cases in which the upper and lower bounds do not match:

- Petri nets and linear-time logics above (and including) LTL. For all these logics, decidability is proved by reduction to the reachability problem. To the best of my knowledge the best upper and lower bounds known for the model-checking problem are just those for the reachability problem, which requires exponential space, but for which no primite recursive algorithm has been given. This complexity gap has remained open for about 15 years.
- BPA-processes and the modal mu-calculus. The best upper bound for the full modal mu-calculus is the one derived from Igo Walukiewicz's algorithm for pushdown processes, which is exponential in both the system and the formula; for the alternation-free mu-calculus, the best upper bound is linear in the size of the system and exponential in the size of the formula. In both cases, there exists no lower bound.

[4] Of those shown in Figure 1, with the trivial exception of Hennesy-Milner logic

- BPPs and linear-time logics above (and including) LTL. The best lower bound for the model-checking problem is PSPACE-hardness, derived from the lower bound for LTL and finite-state systems, while the upper bound is derived from the decidability of the linear-time mu-calculus for Petri nets, and states that the problem is at least as hard as the reachability problem.

There exist also some open problems in the area of local model-checking techiques, in particular tableau methods. Several people (including myself) have searched for a tableau method for BPA (or pushdown processes) and the *full* modal mu-calculus without success so far. In my opinion, we also need better sound and complete tableau methods for undecidable problems: the research initiated by Julian Bradfield in [3] and Hardi Hungar [19] shows the way.

Acknowledgements

Many thanks to Faron Moller for allowing me to use pieces of text from his paper 'Infinite results'.

References

[1] M. Ben-Ari, Z. Manna and A. Pnueli (1983). The temporal logic of branching time. *Acta Informatica* 20(3): 207–226.

[2] J.A. Bergstra and J.W. Klop (1985). Algebra of communicating processes with abstraction. *Theoretical Computer Science* 37:77–121.

[3] J. Bradfield (1991). *Verifying Temporal Properties of Systems*. Birkhäuser, Boston, Massachussets.

[4] A. Bouajjani, R. Echahed, and R. Robbana (1994). Verification of nonregular temporal properties for context-free processes. Proceedings of CONCUR'94. *Lecture Notes in Computer Science* 836: 81–97.

[5] A. Bouajjani, R. Echahed, and P. Habermehl (1995). On the verification problem of nonregular properties for nonregular processes. Proceedings of LICS'95, IEEE.

[6] A. Bouajjani and P. Habermehl (1996). Constrained properties, semilinear systems, and Petri nets. To appear in Proceedings of CONCUR '96.

[7] A. Bouajjani and O. Maler (1996). Reachability analysis of pushdown automata. To appear in Proceedings of INFINITY'96.

[8] O. Burkart, D. Caucal and B. Steffen (1996). Bisimulation collapse and the process taxonomy. To appear in Proceedings of CONCUR'96.

[9] O. Burkart and Y.M. Quemener (1996). Model-checking of infinite graphs defined by graph grammars. To appear in Proceedings of INFINITY'96.

[10] O. Burkart and B. Steffen (1992). Model checking for context-free processes Proceedings of CONCUR'92. *Lecture Notes in Computer Science* 630:123–137.

[11] O. Burkart and B. Steffen (1994). Pushdown Processes: Parallel composition and model-checking. Proceedings of CONCUR'94. *Lecture Notes in Computer Science* 836:98–113.

[12] D. Caucal (1992). On the regular structure of prefix rewriting. *Journal of Theoretical Computer Science* 106:61–86.

[13] D. Caucal (1996). On infinite transition graphs having a decidable monadic theory. Proceedings of ICALP'96.. *Lecture Notes in Computer Science* 1099:194–205.

[14] A. Cheng, J. Esparza, and J. Palsberg (1995). Complexity results for 1-safe Petri nets. *Theoretical Computer Science* 147: 117–136.

[15] S. Christensen (1993). *Decidability and Decomposition in Process Algebras*. Ph.D. Thesis ECS-LFCS-93-278, Department of Computer Science, University of Edinburgh.

[16] J. Esparza (1994). On the decidability of model-checking for several μ-calculi and Petri nets. Proceedings of CAAP'94 *Lecture Notes in Computer Science* 787: 115–129.

[17] J. Esparza (1995). Decidability of model-checking for concurrent infinite-state systems. To appear in *Acta Informatica*.

[18] J. Esparza and A. Kiehn (1995). On the decidability of model-checking for branching time logics and Basic Parallel Processes. Proceedings of CAV'95, *Lecture Notes in Computer Science* 939: 353–366.

[19] H. Hungar (1994). Local model-checking for parallel compositions of context-free processes. Proceedings of CONCUR'94.*Lecture Notes in Computer Science* 836: 114–128.

[20] H. Hungar (1994). Local model-checking of higher-order processes. Technical Report, Universität Oldenburg.

[21] H. Hungar and B. Steffen (1993). Local model-checking for context-free processes. Proceedings of ICALP'93. *Lecture Notes in Computer Science* 700:593–605.

[22] M. Hack (1976). *Decidability questions for Petri nets*. Ph.D. Thesis, Technical Report 161, Laboratory for Computer Science, Massachusetts Institute of Technology.

[23] F. Moller (1996). *Infinite results*. To appear in Proceedings of CONCUR'96.

[24] E. Mayr (1984). An algorithm for the general Petri net reachability problem. *SIAM Journal of Computing* 13:441–460.

[25] R. Mayr (1996). Some results on basic parallel processes. SFB-Report, Technische Universität München.

[26] R. Mayr (1996). A tableau system for model-checking Petri nets with a subset of the linear time mu-calculus. SFB-Report, Technische Universität München.

[27] R. Mayr (1996). Private communication.

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

[29] F. Moller and G. Birtwistle (eds.) (1996). *Logics for Concurrency. Lecture Notes in Computer Science* 1043.

[30] R. Lipton (1976). The reachability problem requires exponential space. Research Report 62, University of Yale.

[31] D. Muller and P. Schupp (1985). The theory of ends, pushdown automata and second order logic. *Theoretical Computer Science* 37:51–75.

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

[33] A.P. Sistla and E.M. Clarke (1985). The complexity of propositional linear temporal logics. *Journal of the ACM* 32(3):733–749.

[34] I. Walukiewicz (1996). Pushdown processes: games and model checking. To appear in Proceedings of CAV'96.

[35] H. Yen (1992). A unified approach for deciding the existence of certain Petri net paths. *Information and Computation* 96(1): 119–137.