URL: http://www.elsevier.nl/locate/entcs/volume6.html ?? pages

# More Infinite Results

Olaf Burkart [a,1] Javier Esparza [b,2]

[a] *LFCS, University of Edinburgh, JCMB, King's Buildings, Edinburgh EH9 3JZ, UK*

[b] *Institut für Informatik, Technische Universität München, Arcisstr. 21, D-80290 München, GERMANY*

**Abstract**

Recently there has been a spurt of activity in concurrency theory centred 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 paper "Infinite Results" [Mol96] , Moller surveys some of the key results on the decidability and complexity of problem (1). This paper is a survey on the results about problem (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?

In his paper "Infinite Results" [Mol96], Moller surveys some of the most relevant results of the first group. He restricts his attention to infinite-state

[1] Supported by the DAAD under grant D/95/14834 of the NATO science committee.
[2] Partially supported by Teilprojekt A3 SAM of the Sonderforschungsbereich 342.

systems having a discrete state space. The purpose of this paper is to give a similar overview of the results of the second group. [3]

The model-checking problem has two parameters: a family of systems, and a temporal logic used to express properties of their behaviour. Following Moller [Mol96] (who takes this idea from Caucal), we consider families of transition systems defined by rewrite rules including sequential rewrite transition systems, like pushdown automata and Basic Process Algebra 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

Concurrent systems can be specified using a variety of formalisms, amongst them algebraic equational specifications, process calculi or Petri nets. Their behaviour is, however, often defined extensionally in a uniform way by the set of all its possible state-transitions together with the observable events that these transitions produce. A simple fundamental model of computation which captures naturally this interpretation is that of rooted labelled transition systems. These are rooted edge-labelled directed graphs where the vertices are interpreted as possible system states, the root denotes the initial state the system starts from, and labelled arcs represent state-transitions of the system where the label corresponds to the event which results upon their execution. Formally, a rooted labelled transition system is defined as follows.

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

- $S$ *is a set of* states.
- $\Sigma$ *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.

While formal language theory considers only the set of words obtained by concatenating the transition labels occurring on a (finite) path from the start state to a terminating state, i.e. a state where no further transitions are possible, process theory is more interested in the structure of the graph itself. As we will see, it is, nevertheless, possible to establish a link between well-studied classes of formal languages and certain classes of transition systems which are uniformly defined by *word rewrite systems*.

**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.

[3] Notice that the equivalence and model-checking problems for dense state spaces are also being actively investigated. Dense state spaces arise naturally in the verification of real-time systems.

- $\Sigma$ *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.

The idea of describing infinite-state processes by means of word rewrite systems where only the restricted form of prefix rewriting is allowed was introduced by Caucal in [Cau92]. A natural extension of his approach is to consider also labelled rewrite transition systems where words are read modulo commutativity of catenation. As in this case the ordering of variables is irrelevant, catenation may be interpreted as parallel, rather than sequential composition.

**Definition 2.3** *A* parallel labelled rewrite transition system *is defined precisely as in Definition 2.2, except that the elements of* $V^*$ *are read modulo commutativity of catenation* [4] .

A number of important families of transition systems, which have been defined so far using other frameworks, can now be characterized uniformly in terms of rewrite systems with restricted kinds of rules. This classification gives raise to a taxonomy of families of infinite-state transition systems which is very similar to the Chomsky hierarchy known from language theory [5] . It is given as follows.

| | Restriction on the rules $\alpha \stackrel{a}{\longrightarrow} \beta$ 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 well-known class of *finite-state automata*, since the condition on the form of allowed rules corresponds directly to right-linearity of grammars in language theory, as well as to the prefix operator of e.g. CCS in process theory. In this case the state set of the generated transition system is

[4] Another point of view is to interprete a process as a multiset where catenation of substrings then corresponds to multiset union.

[5] It is, however, not clear at the moment whether the class of transition systems defined by *context-sensitive* rules with the condition $|\alpha| \leq |\beta|$ have any interesting properties when interpreted within process theory.

therefore clearly finite, and consists, more precisely, of the subset of variables which is reachable from the initial state.

Adding unrestricted sequential composition to FSA yields the class of Basic Process Algebra (**BPA**) processes introduced by Bergstra and Klop [BK85]. As these processes correspond to context-free grammars in Greibach normal form (GNF) where only left-most derivations are permitted, they are also called *context-free processes*. Typically, BPA processes allow to model some stack-like behaviour, which may degenerate down to the infinite binary tree, as depicted in Figure 1.

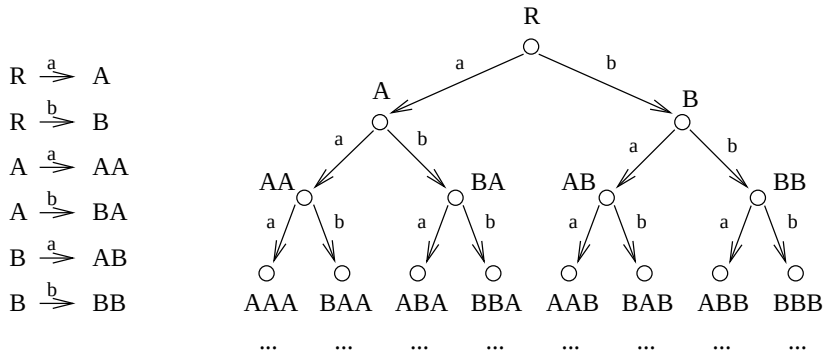


Fig. 1. The infinite binary tree as a BPA process.

The next step up in the sequential hierarchy leads to the class of sequential Type 0 systems. Caucal [Cau92] has shown that for any sequential Type 0 system $\mathcal{S}$ there exists a *pushdown automaton* (**PDA**) $\mathcal{A}$ accepting on empty stack such that the transition systems generated by $\mathcal{S}$ and $\mathcal{A}$ are isomorphic up to relabelling. Due to this result, we identify sequential Type 0 systems and PDAs. Caucal and Monfort have shown that this result does not hold for BPA processes [CM90]: it is possible to construct a sequential Type-0 system $\mathcal{S}$ such that no transition system generated by BPA processes is isomorphic, or even bisimilar to the transition system of $\mathcal{S}$ . Notice the contrast with the classical result of language theory stating that the class of languages generated by context-free grammars coincides with the class of languages accepted by pushdown automata.

Still another characterization of the sequential Type-0 was (indirectly) given in [BS95]: it is shown there that the class of PDA transition systems is exactly the closure of BPA transition systems under synchronized parallel composition with finite-state systems.

In the remainder of the paper we use the classical representation for PDAs for sequential Type 0 systems, where the variable set $V$ is partitioned into disjoint sets $Q$ (finite control states) and $\Gamma$ (stack symbols), and where rewrite rules are of the form $pA \stackrel{a}{\longrightarrow} q\beta$ with $p, q \in Q$, $A \in \Gamma$ and $\beta \in \Gamma^*$.

The first interesting class of infinite processes in the parallel branch of the hierarchy is the class of Basic Parallel Processes (**BPP**) introduced by

Christensen [Chr93]. They are a parallel analogy to BPA, and correspond to the transition systems generated by GNF context-free grammars in which arbitrary derivations are permitted, not only leftmost ones, as in the case of BPA. BPP languages are a subclass of the context-sensitive languages which is incomparable with the class of context-free languages. On the other hand, BPP processes typically generate grid-like transition systems which in its purest form yields the infinite grid as depicted in Figure 2.

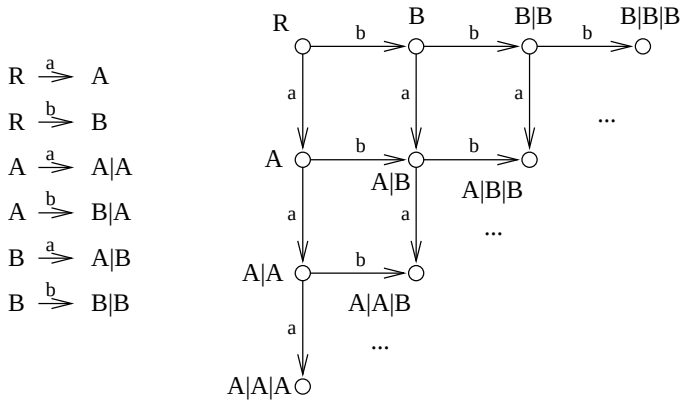


Fig. 2. The infinite grid as a BPP process.

**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 $\alpha$ and $\beta$ respectively, with the weights on the input and output arcs given by the relevant multiplicities in $\alpha$ and $\beta$. Note that a BPP is a Petri net in which each transition has a unique input place, and the weight of the arc from the place to the transition is 1.

Summarizing, the process classes introduced so far can be classified according to their expressiveness as depicted in Figure 3. Examples of processes which separate these classes can be found in [Mol96].

## 3 Temporal logics

In the model-checking approach to verification, temporal logics are used to express properties of transition systems: a rewrite system $R$ satisfies a property $P$ if the transition system of $R$ is a model of the temporal formula corresponding to $P$.

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

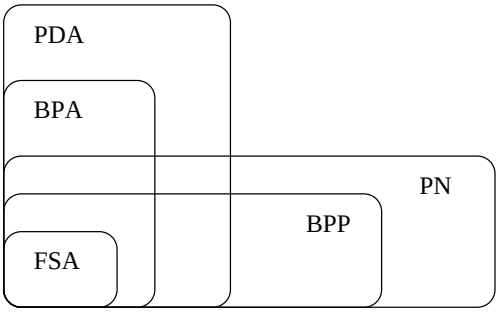


Fig. 3. A taxonomy of classes of infinite-state transition systems.

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 4 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 since they are very popular, although strictly speaking they do not play a rôle in this paper.

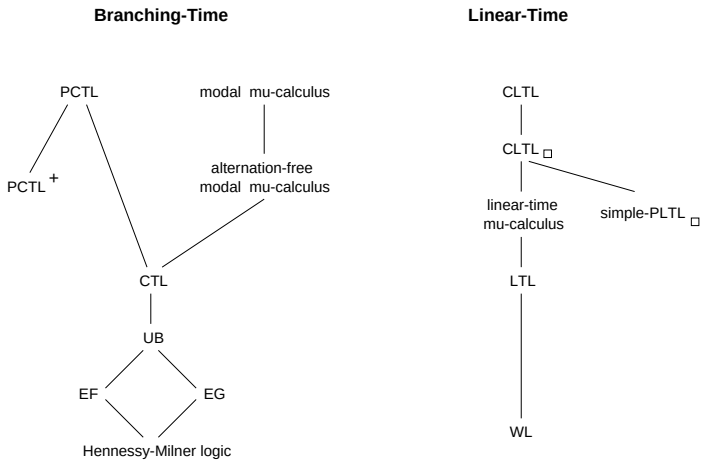


Fig. 4. Linear and branching-time logics.

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

for instance, in the contributions of E. Allan Emerson and Colin Stirling to [MB96]. 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 $s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \ldots$ such that every triple $s_i \xrightarrow{a_i} s_{i+1}$ 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 then interpreted on the runs of a transition system, and 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 in this paper, 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 $\phi$.

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

The *linear-time mu-calculus* adds variables to WL, together with a 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. [6]

The most expressive linear-time logic is, finally, *Constrained Linear Temporal Logic* (CLTL), which was introduced in [BEH95]. It allows to define nonregular properties, i.e. properties which are nondefinable by finite-state $\omega$-automata. This expressiveness is achieved by extending 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. Two important fragments of CLTL which still allow to specify a wide range of nonregular properties are CLTL$_\Box$, where counting constraints cannot be introduced in eventuality formulae, and simple-PLTL$_\Box$, the extension of LTL with counting constraints, where only propositional formulae may be used in eventuality formulae. CLTL$_\Box$, and therefore CLTL, is strictly more expressive than the linear-time mu-calculus, since e.g. the typical property of runs that "every finite prefix contains at least as many $a$ actions as $b$ actions" is expressible in CLTL$_\Box$ but not in the linear-time mu-calculus. On the other hand, simple-PLTL$_\Box$ is incomparable to LTL, but allows to express the complement of simple $\omega$-regular languages.

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

**Branching-time logics**

A formula of a branching-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 $E(a)\phi$ ($A(a)\phi$), which holds at a state $s$ if some run (all runs) starting at $s$ satisfy $(a)\phi$.[7] Notice that $A(a)\phi = \neg E(a)\neg\phi$.

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

The *Unified System of Branching-Time Logic* (UB) [BAMP83] adds to Hennessy-Milner logic the operators $EF\phi$ and $EG\phi$. The fragments of $UB$ containing only the operator $EF\phi$ ($EG\phi$) is called $EF$ ($EG$) in this paper.

*Computation Tree Logic* (CTL) adds to Hennessy-Milner logic an *until* operator $E\phi\,U\,\psi$, and so it can be considered the branching-time counterpart of LTL.

*Presburger Computation Tree Logic* (PCTL) [BER94] extends CTL with counting constraints on actions expressed by Presburger arithmetic formulae. The fragment PCTL$^+$ is obtained by imposing the following syntactic restriction: in every subformulae of the form $E\phi\,U\,\psi$, the formula $\psi$ is a counting constraint, which in particular contains no temporal operators.

The *modal mu-calculus* extends Hennessy-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 $\phi$. 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 $\nu$-subformula has a free variable which, in the context of the whole formula, is bound by a $\mu$, and vice versa. 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 [MS85] it is shown that the monadic second order logic (MSOL) on the class of transition systems generated by pushdown automata is decidable; since the modal mu-calculus can be strictly embedded into monadic second order logic, the decidability of the modal mu-calculus follows. The algorithm derived from this decidability proof is, however, based on the decidability of MSOL for the infinite binary

[7] 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 Hennessy-Milner logic.

tree established by Rabin [Rab69], and therefore extremely inefficient: it has non-primitive recursive complexity.

The first algorithm which tried to overcome this problem was given by the first author and Steffen in [BS95], where a global model-checking algorithm for the alternation-free modal mu-calculus is presented. The algorithm is an extension of a similar algorithm for BPA processes, which is discussed in the next section.

Recently, Walukiewicz has improved on these results by presenting in [Wal96] a model-checking algorithm based on games. He shows, in particular, that there exists an alternation-free mu-calculus formula $\Phi$ such that the problem "given a PDA $\mathcal{A}$, does the start configuration of $\mathcal{A}$ satisfy $\Phi$" is DEXPTIME-complete. Hardness is proved by reduction from the problem of deciding if an alternating linear space bounded Turing machine accepts a given input, while membership in DEXPTIME is shown 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 by 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.

Notice that the model-checking problem is DEXPTIME-complete *in the size of the system*, because it is DEXPTIME-complete for a fixed formula $\Phi$.

A different approach to model-checking pushdown automata was pursued by Bouajjani and Maler in [BM96], where they show that the set of configurations satisfying a CTL property can be recognized essentially by a finite alternating multi automaton. The main technique they rely on is the ability to compute effectively the set of predecessors of a given set of states in terms of finite alternating automata. This approach is then inductively applied to the CTL formula of interest, as the automaton corresponding to a formula can be computed from the automata corresponding to its subformulae. [8]

### Linear-time logics

The decidability of the linear-time mu-calculus for pushdown automata follows from the decidability of the modal mu-calculus, but can also be easily

[8] The method has been recently extended to the alternation-free mu-calculus [BE96].

established using the automata-theoretic approach to model-checking. Given a formula $\phi$ of the linear-time mu-calculus, we build a Büchi automaton $A_{\neg\phi}$ which accepts all computations that do not satisfy $\phi$. Then, we construct the product of $A_{\neg\phi}$ and the pushdown automaton to yield an $\omega$-pushdown automaton $A_P$. The formula $\phi$ holds if and only if $A_P$ is empty, a decidable problem. In fact, this algorithm shows that the model-checking problem can be solved in DEXPTIME. Recently, Bouajjani and the second author have shown that the problem is DEXPTIME-hard [BE96].

Bouajjani and Maler have generalized this approach in [BM96] by using, as in the case of model-checking CTL, finite alternating multi-automata to characterize the set of *all* states of a pushdown transition system that satisfies a given $\omega$-regular property.

Finally, decidability of nonregular linear time properties has been investigated by Bouajjani and Habermehl in [BH96]. They prove that $\mathrm{CLTL}_\Box$ is decidable for pushdown automata which is a remarkable result, as full CLTL is undecidable even for finite-state systems. The proof consists of two steps:

(1) It is shown that any formula of $\mathrm{CLTL}_\Diamond$, the complement of $\mathrm{CLTL}_\Box$, can be transformed into essentially a conjunction of a formula of the linear-time mu-calculus, which expresses an $\omega$-regular property, and a pure counting constraint.

(2) It is shown that the model-checking problem for counting constraints is decidable whenever the intersection of the $\omega$-regular property with the $\omega$-language generated by the pushdown automaton is semilinear. As it is known that pushdown transition systems generate $\omega$-context-free languages, which are closed under intersection with $\omega$-regular languages and semilinear, the model-checking problem for $\mathrm{CLTL}_\Diamond$ is decidable, and consequently so is the model-checking problem for $\mathrm{CLTL}_\Box$.

### *4.2 Petri Nets*

While all the logics of Figure 4 below and including the modal mu-calculus were decidable for pushdown automata, 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, Hennessy-Milner logic) become undecidable. A survey of these results has been given by the second author in [Esp95], and most of what follows is taken from there.

#### Branching-time logics

The undecidability of nearly all the branching-time logics in Figure 4 is shown by reduction from the halting problem for Minsky machines [Min67] (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 [Mol96]). The following description of Minsky machines is taken verbatim from [Mol96].

Minsky Machines [Min67] 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

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

or of the form

$\mathtt{X}_\ell : \mathtt{if}\ \mathtt{c}_0 = 0\ \mathtt{then}\ \mathtt{goto}\ \mathtt{X}_j\ \mathtt{else}\ \mathtt{c}_0 := \mathtt{c}_0{-}1;\ \mathtt{goto}\ \mathtt{X}_k$ or $\mathtt{X}_\ell : \mathtt{if}\ \mathtt{c}_1 = 0\ \mathtt{then}\ \mathtt{goto}\ \mathtt{X}_j\ \mathtt{else}\ \mathtt{c}_1 := \mathtt{c}_1{-}1;\ \mathtt{goto}\ \mathtt{X}_k$

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 $\mathrm{comm}_\ell$, 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, since Minsky machines are Turing powerful and Petri nets are not [Pet81]. 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 4 above Hennessy-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 4, 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 [Hac76], by means of a rather involved reduction from Hilbert's tenth problem.)

### Linear-time logics

Since the only atomic sentence of our linear-time temporal logics is *true*, they 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 reduction to the halting problem for Minsky machines that was used to prove the undecidability of branching-time logics does not work anymore. In fact, it turns out that all the linear-time temporal logics of Figure 4 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 [Esp94] using the automata-theoretic approach. As before, given a formula $\phi$ of the linear-time mu-calculus, we build a Büchi automaton $A_{\neg\phi}$ which accepts all computations that do not satisfy $\phi$. 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 $\phi$ 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 [Yen92]; (1) can be reduced to the reachability problem, which is decidable [May84] and requires at least exponential space [Lip76], but for whose complexity no primitive recursive upper bound has been found so far. Unfortunately, (1) cannot be reduced to any simpler problem because the model-checking problem for the linear-time mu-calculus is at least as hard as reachability. [9]

It is however possible to identify a fragment of the linear-time mu-calculus which can be reduced to (2) only. For this fragment, Mayr has recently obtained a decidable tableau-system [May96c].

Bouajjani and Habermehl show in [BH96] that the model-checking problem for $\mathrm{CLTL}_{\Box}$ is decidable. 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 automata. This time, the result is proved by means of a reduction to the reachability problem.

[9] This can be shown in different ways. For instance, the linear-time mu-calculus can express deadlock-freeness, and the reachability problem for Petri nets can be reduced in polynomial time to the deadlock-freeness problem [CEP95].

# 5 Results for Type 2 Rewrite Systems

In this section only branching-time logics are dealt with in detail, 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 as well; and,
- Peter Habermehl and the second author have recently shown that the model-checking problem for BPPs and LTL is EXPSPACE-hard [HE96]. Essentially, this shows that model-checking BPPs is as hard as model-checking general Petri nets.

## *5.1 Processes of Basic Process Algebra*

Caucal and Monfort observed in [CM90] that pushdown automata are strictly more expressive than BPA processes with respect to bisimulation equivalence. It may therefore well be the case that there exist simpler model-checking algorithms for BPA processes than for pushdown automata. So far, this question remains, however, open for the modal mu-calculus, as the complexity of the model-checking problem for BPA processes and the modal mu-calculus has not been established yet.

The first author and Steffen present in [BS92] 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 Walukiewicz, all algorithms for the same logic and pushdown automata need *exponential* time in the size of the system. So, for the alternation-free mu-calculus, moving down from pushdown automata to BPA processes does indeed make model-checking easier.

Although the original algorithm of [BS92] works on formulations of BPA processes and alternation-free formulas which are rather different from those used in this paper, an adaptation to our setting is straightforward. The model-checking algorithm proceeds by iteratively computing a *property transformer* for each nonterminal of the given BPA system. A property transformer for a nonterminal $A$ takes a set of formulae $\Delta$ as arguments, and yields the set of formulas valid at $A$ under the assumption that all formulas of $\Delta$ are valid after termination of $A$, i.e. at the final state $\varepsilon$ from which no action can occur. Once the property transformers have been computed, the model-checking problem is solved by taking the property transformer of the nonterminal corresponding to the initial state of the BPA process, and applying it to the set of formulae satisfied by $\varepsilon$ (which can be easily computed using a standard finite-state model-checking algorithm, since it is a deadlocked state).

This model-checking algorithm is *global*, as it provides complete information about the formulae satisfied by all the states of the BPA process under consideration. In [HS93], Hungar and Steffen present a local model-checking alternative to [BS92] in the form of a remarkably simple tableau system. The sequents of the tableaux are again inspired by the notion of property trans-

former. They are of the form $A \vdash \langle \phi, \Delta \rangle$, with intended meaning $\phi$ holds at state $A$ under the assumption that all formulas of $\Delta$ hold at the final state. The heart of the tableau system is the following composition rule, clearly inspired by the sequential composition rule of Hoare logic:

$$\frac{\alpha\beta \vdash \langle \phi, \Delta \rangle}{\alpha \vdash \langle \phi, \Gamma \rangle \quad \beta \vdash \langle \Gamma, \Delta \rangle}$$

The rule has the following intended meaning. In order to know if $\phi$ holds at $\alpha\beta$ assuming that $\Delta$ holds at the final state, we guess an *intermediate assertion* $\Gamma$, and prove the following:

- if $\Delta$ holds at the final state, then $\Gamma$ holds before execution of $\beta$;
- if $\Gamma$ holds after execution of $\alpha$, then $\phi$ holds immediately before its execution.

Finally, we mention that the verification of nonregular properties for BPA processes has been considered by Bouajjani, Echahed and Robbana in [BER94]. They show that the logic PCTL is undecidable even for finite-state systems. Weakening, subsequently, PCTL to its fragment PCTL$^+$ they obtain on the other hand decidability by reduction to the validity problem of Presburger arithmetic.

### *5.2 Basic Parallel Processes*

**Branching-time logics**

In [EK95], Kiehn and the second 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, as (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. It is still possible to characterise the 'honest runs' of the system in the logic $EG$, although the formula becomes very complicated.

After this result, the only branching-time logic of Figure 4 which remains to be explored is $EF$. It was shown in [Esp95] that the model-checking problem for this logic is PSPACE-hard even for *finite* BPPs (BPP may encode finite-state systems much more succinctly than finite automata). Mayr has recently shown that the problem is PSPACE-complete [May96b], 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 (in the size of $N$ and $(n_1, \ldots, n_k)$) if $N \xrightarrow{w}$ for some sequence of labels $w$ containing $n_i$ times each label $a_i$;

- if a BPP 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 $\phi$.

## 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. The main aim of this research has been to obtain a better understanding of the border between decidability and undecidability, as well as the underlying structure of proofs, in general.

### Extensions of BPA processes

Hungar has extended the tableau system of [HS93] for BPA processes and the alternation-free modal mu-calculus in two different directions:

In [Hun94a], 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, as the parallel composition of two BPA processes (with communication) can simulate a Turing Machine. It is, however, shown to be decidable for the special case in which at most one of the communicating processes is infinite-state. This latter result coincides with the observation of Burkart and Steffen in [BS95] that the synchronous parallel composition of a BPA process with a finite-state process is essentially a pushdown automaton for which the model-checking problem is known to be decidable.

In [Hun94b], he introduces *macro processes*. This framework extends the interpretation of BPA processes as procedures, first given in [BS92], by allowing transitions with "higher-order procedure calls". The idea is that a nonterminal corresponds to a procedure where its right-hand side represents the body of the procedure. Macro processes add to this simple model the possibility to use typed parameters in procedure calls which are again procedure calls (of a smaller type). For these processes a local, as well as a global, iterative model-checking algorithm for the alternation-free mu-calculus are given which have $k$-exponential complexity where $k$ is the depth of the type hierarchy.

### Extensions of pushdown automata

Besides the rewrite approach we have emphasized in this survey, infinite transition systems may also be described in terms of other devices. An attractive candidate is the framework of deterministic graph grammars where nonterminals are replaced by hyperedges, and right-hand side words by finite graphs which may again contain hyperedges. Graph grammars characterize the class of *regular graphs* which is strictly more expressive than the class of pushdown automata as they also allow for infinite-branching [Cau92]. Nevertheless, Courcelle has shown that they still possess a decidable monadic second

order theory [Cou90]. Recently, the first author and Quemener have presented at INFINITY'96 [BQ96] a model-checking algorithm for the alternation-free mu-calculus which improves on the former nonelementary complexity obtained by the mere decidability result of Courcelle. They extend the method of computing property transformer for nonterminals [BS92,BS95] to whole finite graphs which may contain hyperedges. The complexity of their algorithm is the same as the complexity of model-checking pushdown automata.

A still more expressive class of transition systems was introduced by Caucal in [Cau96]. He considers the class of infinite transition graphs $\mathrm{REC}_{Rat}$ defined by rewrite systems where in rewrite rules the left-hand side, as well as the right-hand side may be regular languages. These rules are interpreted as the infinite union of "ordinary" rules $\alpha \stackrel{a}{\longrightarrow} \beta$ where $\alpha$ is a word of the regular language on the left-hand side, while $\beta$ is a word of the regular language on the right-hand side, respectively. Since these rules may only be applied wrt. prefix rewriting, $\mathrm{REC}_{Rat}$ belongs to an extension of sequential rewrite transition systems in which the set of rules may be infinite. His main results are that this class properly extends the class of regular graphs, and that $\mathrm{REC}_{Rat}$ has a decidable monadic second order theory.

### 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. As the transition systems of this class of processes properly contains all those generated by Type 2 rewrite transition systems, and therefore, in particular those of BPP, the only logic of Figure 4 that might still be decidable for PA is the branching-time logic $EF$. Mayr has very recently shown that this is indeed the case [May96a].

Concerning linear-time properties, Bouajjani and Habermehl observe in [BH96] 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 $\omega$-star-free language (which corresponds to some LTL formula) and a PA language. Nevertheless, they were able to show that the verification problem of PA wrt. the weaker logic simple-$\mathrm{PLTL}_{\Box}$ is decidable.

### Arbitrary systems

Bradfield has proposed in [Bra91] 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.

Gurov, Berezin, and Kapron have proposed a sound tableau system for a very powerful fist order modal mu-calculus and arbitrary value-passing CCS processes [GBK96]. The tableau is complete for certain fragments of the logic.

# 7 Conclusions

This paper has surveyed work on the model-checking problem for infinite-state systems. The many results cited in the paper can be condensed into three main conclusions:

*Most branching-time and linear-time logics are decidable for sequential Type 0 systems, whereas 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 [10] 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 Type 0 systems is in DEXPTIME.
- The model-checking problem for LTL and parallel Type 2 systems is EXPSPACE-hard.

The third (and for the authors 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 following results:

- none of the logics of Figure 4 is undecidable for PDAs and decidable for BPAs.
- $EF$ is the only logic of Figure 4 that is undecidable for Petri nets and decidable 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 in our opinion rather 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 our 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 primitive recursive algorithm has been given. This complexity gap has remained open for about 15 years.

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

- BPA processes and the modal mu-calculus. The best upper bound for the full modal mu-calculus is the one derived from Igor Walukiewicz's algorithm for pushdown automata, 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.

There exist also some open problems in the area of local model-checking techniques, in particular tableau methods. Several people (including the second author) have searched for a tableau method for BPA (or pushdown automata) and the *full* modal mu-calculus, without success so far. Better sound and complete tableau methods for undecidable problems are also needed: the research initiated in [Bra91,Hun94a,GBK96] shows the way.

*Acknowledgements*

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

## References

[BAMP83] M. Ben-Ari, Z. Manna, and A. Pnueli. The Temporal Logic of Branching Time. *Acta Informatica*, 20(3):207–226, 1983.

[BEH95] R. Bouajjani, R. Echahed, and P. Habermehl. On the Verification of Nonregular Properties for Nonregular Processes. In *LICS '95*, pages 123–133. IEEE Computer Society Press, 1995.

[BER94] A. Bouajjani, R. Echahed, and R. Robbana. Verification of Nonregular Temporal Properties for Context-Free Processes. In *CONCUR '94*, LNCS 836, pages 81–97. Springer, 1994.

[BE96] A. Bouajjani and J. Esparza. Private communication. 1996.

[BH96] A. Bouajjani and P. Habermehl. Constrained Properties, Semilinear Systems, and Petri Nets. In *CONCUR '96*, LNCS 1119, pages 481–497. Springer, 1996.

[BK85] J.A. Bergstra and J.W. Klop. Algebra of Communicating Processes with Abstraction. *Theoretical Computer Science*, 37:77–121, 1985.

[BM96] A. Bouajjani and O. Maler. Reachability Analysis of Pushdown Automata. In *INFINITY '96*, MIP-9614, pages 99–114. Universität Passau, July 1996.

[BQ96] O. Burkart and Y.-M. Quemener. Model-Checking of Infinite Graphs Defined by Graph Grammars. In *INFINITY '96*, MIP-9614, pages 56–70. Universität Passau, July 1996.

[Bra91] J.C. Bradfield. *Verifying Temporal Properties of Systems.* Birkhäuser, Boston, Massachussets, 1991.

[BS92] O. Burkart and B. Steffen. Model Checking for Context-Free Processes. In *CONCUR '92*, LNCS 630, pages 123–137. Springer, 1992.

[BS95] O. Burkart and B. Steffen. Composition, Decomposition and Model-Checking of Pushdown Processes. *Nordic Journal of Computing*, 2:89–125, 1995.

[Cau92] D. Caucal. On the Regular Structure of Prefix Rewriting. *Theoretical Computer Science*, 106:61–86, 1992.

[Cau96] D. Caucal. On Infinite Transition Graphs Having a Decidable Monadic Theory. In *ICALP '96*, LNCS 1099, pages 194–205. Springer, 1996.

[CEP95] A. Cheng, J. Esparza, and J. Palsberg. Complexity Results for 1-Safe Petri Nets. *Theoretical Computer Science*, 147:117–136, 1995.

[Chr93] S. Christensen. *Decidability and Decomposition in Process Algebras.* PhD thesis, The University of Edinburgh, Department of Computer Science, 1993.

[CM90] D. Caucal and R. Monfort. On the Transition Graphs of Automata and Grammars. In *Graph-Theoretic Concepts in Computer Science*, LNCS 484, pages 311–337. Springer, 1990.

[Cou90] B. Courcelle. Graph Rewriting: An Algebraic and Logic Approach. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 5, pages 193–242. Elsevier Science Publisher B.V., 1990.

[EK95] J. Esparza and A. Kiehn. On the Model Checking Problem for Branching Time Logics and Basic Parallel Processes. In *CAV '95*, LNCS 939, pages 353–366. Springer, 1995.

[Esp94] J. Esparza. On the Decidability of Model Checking for Several $\mu$-calculi and Petri Nets. In *CAAP '94*, LNCS 787, pages 115–129. Springer, 1994.

[Esp95] J. Esparza. Decidability of Model-Checking for Concurrent Infinite-State Systems. To appear in Acta Informatica, 1995.

[GBK96] D. Gurov, S. Berezin, and B.M. Kapron. A Modal $\mu$-Calculus and a Proof System for Value Passing Processes. In *INFINITY '96*, MIP-9614, pages 99–114. Universität Passau, July 1996.

[HE96] P. Habermehl and J. Esparza. Private communication. 1996

[Hac76] M. Hack. *Decidability Questions for Petri Nets.* PhD thesis, Laboratory for Computer Science, Massachusetts Institute of Technology, 1976.

[HS93] H. Hungar and B. Steffen. Local Model-Checking for Context-Free Processes. In *ICALP '93*, LNCS 700, pages 593–605, 1993.

[Hun94a] H. Hungar. Local Model Checking for Parallel Compositions of Context-Free Processes. In *CONCUR '94*, LNCS 836, pages 114–128, 1994.

[Hun94b] H. Hungar. Model-Checking of Macro Processes. In *CAV '94*, LNCS 818, pages 169–181. Springer, 1994.

[Lip76] R. Lipton. The Reachability Problem Requires Exponential Space. Research report 62, University of Yale, 1976.

[May84] E. Mayr. An Algorithm for the General Petri Net Reachability Problem. *SIAM Journal of Computing*, 13:441–460, 1984.

[May96a] R. Mayr. Private communication. 1996.

[May96b] R. Mayr. Some Results on Basic Parallel Processes. SFB-Report, Technische Universität München, 1996.

[May96c] R. Mayr. A Tableau System for Model-Checking Petri Nets with a subset of the Linear Time Mu-Calculus. SFB-Report, Technische Universität München, 1996.

[MB96] F. Moller and G. Birtwistle, editors. *Logics for Concurrency.* LNCS 1043. Springer, 1996.

[Min67] M. Minski. *Computation: Finite and Infinite Machines.* Prentice-Hall, 1967.

[Mol96] F. Moller. Infinite Results. In *CONCUR '96*, LNCS 1119, pages 195–216. Springer, 1996.

[MS85] D.E. Muller and P.E. Schupp. The Theory of Ends, Pushdown Automata, and Second-Order Logic. *Theoretical Computer Science*, 37:51–75, 1985.

[Pet81] J.L. Peterson. *Petri Net Theory and the Modelling of Systems.* Prentice-Hall, 1981.

[Rab69] R.O. Rabin. Decidability of Second-Order Theories and Automata on Infinite Trees. *Transactions of the AMS*, 141:1–35, 1969.

[SC85] A.P. Sistla and E.M. Clarke. The Complexity of Propositional Linear Temporal Logics. *Journal of the ACM*, 32(3):733–749, 1985.

[Wal96] I. Walukiewicz. Pushdown Processes: Games and Model-Checking. In *CAV '96*, LNCS 1102. Springer, 1996.

[Yen92] H. Yen. A Unified Approach for Deciding the Existence of Certain Petri Net Paths. *Information and Computation*, 96(1):119–137, 1992.