Verification on Infinite Structures

Olaf Burkart

Lehrstuhl Informatik V
Universität Dortmund
Baroper Straße 301
D-442 21 Dortmund
GERMANY

<burkart@cs.uni-dortmund.de>

Didier Caucal

IRISA
Campus de Beaulieu
F-350 42 Rennes
FRANCE

<caucal@irisa.fr>

Faron Moller

Dept of Computer Science
University of Wales Swansea
Singleton Park, Sketty
Swansea SA2 8PP
WALES

<F.G.Moller@swansea.ac.uk>

Bernhard Steffen

Lehrstuhl Informatik V
Universität Dortmund
Baroper Straße 301
D-442 21 Dortmund
GERMANY

<steffen@cs.uni-dortmund.de>

Abstract. In this chapter, we present a hierarchy of infinite-state systems based on the primitive operations of sequential and parallel composition; the hierarchy includes a variety of commonly-studied classes of systems such as context-free and pushdown automata, and Petri net processes. We then examine the equivalence and regularity checking problems for these classes, with special emphasis on bisimulation equivalence, stressing the structural techniques which have been devised for solving these problems. Finally, we explore the model checking problem over these classes with respect to various linear- and branching-time temporal logics.

Keywords: infinite-state rewrite transition systems, sequential/parallel composition/computation, automatic verification, decidability, complexity, behavioural equivalences, bisimulation, equivalence checking, regularity checking, linear/branching-time temporal logics, model checking

Contents

Introduction **3**

1 A Taxonomy of Infinite State Processes **8**
1.1 Rewrite Transition Systems 8
1.2 Languages and Bisimilarity 14
1.3 Expressivity Results 19
1.4 Further Classes of Processes 23

2 The Equivalence Checking Problem **25**
2.1 Decidability Results for BPA and BPP 25
2.1.1 Composition and Decomposition 25
2.1.2 A Finite Bisimulation Base for BPA 29
2.1.3 A Finite Bisimulation Base for BPP 31
2.2 Polynomial Time Algorithms for BPA and BPP 32
2.2.1 A Polynomial Time Algorithm for Normed BPA 33
2.2.2 A Polynomial Time Algorithm for Normed BPP 36
2.3 Computing a Finite Bisimulation Base for BPA 38
2.4 Undecidability Results for MSA 47
2.5 Summary of Results 50

3 The Model Checking Problem **53**
3.1 Temporal Logics 53
3.1.1 Branching-Time Logics 56
3.1.2 Linear-Time Logics 61
3.2 Algorithms for Branching-Time Logics 65
3.2.1 BPA 65
3.2.2 Pushdown Processes and Extensions 69
3.2.3 BPP and Petri Nets 72
3.2.4 PA 73
3.2.5 General Processes 73
3.3 Algorithms for Linear-Time Logics 74
3.3.1 BPA and Pushdown Processes 75
3.3.2 BPP and Petri Nets 76
3.3.3 PA 79
3.4 Summary 79

Introduction

The study of automated (sequential) program verification has an inherent theoretical barrier in the guise of the halting problem and formal undecidability. The simplest programs which manipulate the simplest infinite data types such as integer variables immediately fall foul of these theoretical limitations. During execution, such a program may evolve into any of an infinitude of states, and knowing if the execution of the program will lead to any particular state, such as the halting state, will in general be impossible to determine. However, this has not prevented a very successful attack on the problem of proving program correctness, and there are now elegant and accepted techniques for the semantic analysis of software.

The history behind the modelling of concurrent systems, in particular hardware systems, has followed a different course. Here, systems have been modelled strictly as finite-state systems, and formal analysis tools have been developed for completely and explicitly exploring the reachable states of any given system, for instance with the goal of detecting whether or not a halting, i.e., deadlocked, state is accessible. This abstraction has been warranted up to a point. Real hardware components are indeed finite entities, and protocols typically behave in a regular fashion irrespective of the diversity of messages which they may be designed to deliver.

In specifying concurrent systems, it is not typical to explicitly present the state spaces of the various components, for example by listing out the states and transition function, but rather to specify them using some higher-level modelling language. Such formalisms for describing concurrent systems are not usually so restrictive in their expressive power. For example, the typical process algebra can encode integer registers, and with them compute arbitrary computable functions; and Petri nets constitute a graphical language for finitely presenting typically infinite-state systems. However, tools which employ such formalisms generally rely on techniques for first assuring that the state space of the system being specified is semantically, if not syntactically, finite. For example, a given process algebra tool might syntactically check that no static operators such as parallel composition appear within the scope of a recursive definition; and a given Petri net tool might check that a net is safe, that is, that no place may acquire more than one token. Having verified the finiteness of the system at hand, the search algorithm can, at least in principle, proceed.

The problem with the blind search approach, which has thwarted attempts to provide practical verification tools, is of course that of state space explosion. The number of reachable states of a system will typically be on the order of exponential in the number of components which make up the system. Hence a great deal of research effort has been expended on taming this state space, typically by developing intelligent search strategies. Various promising techniques have been developed which make for the automated analysis of extremely large state spaces feasible; one popular approach to this problem is through the use of BDD (binary decision diagram) encodings of automata [20]. However, such approaches are inherently bound to the analysis of finite-state systems.

Recently, interest in addressing the problem of analysing infinite-state sys-

tems has blossomed within the concurrency theory community. The practical motivation for this has been both to provide for the study of parallel program verification, where infinite data types are manipulated, as well as to allow for more faithful representations of concurrent systems. For example, real-time and probabilistic models have come into vogue during the last decade to reflect for instance the temporal and nondeterministic behaviour of asynchronous hardware components responding to continuously-changing analogue signals; and models have been developed which allow for the dynamic reconfiguration of a system's structure. Such enhancements to the expressive power of a modelling language immediately give rise to infinite-state models, and new paradigms not based on state space search need to be introduced to successfully analyse systems expressed in such formalisms.

In this survey, we explore the two major streams in system verification: ***equivalence checking***, which aims at establishing some semantic equivalence between two systems, one of which is typically considered to represent the implementation of the specification given by the other; and ***model checking***, which aims at determining whether or not a given system satisfies some property which is typically presented in some modal or temporal logic. In the survey we concentrate exclusively on discrete systems, and neglect infinite-state structures related to, e.g., timed and hybrid systems. The variety of process classes which we consider is catalogued in Section 1, where we detail the relative expressive powers of these classes. The two approaches to system verification are summarized here, where we also sketch the contents of the two relevant sections of the survey.

Equivalence Checking

Equivalence checking, that is, determining when two (infinite-state) systems are in some semantic sense equal, is clearly a particularly relevant problem in system verification. Indeed, such questions have a long tradition in the field of (theoretical) computer science. Since the proof by Moore [121] in 1956 of the decidability of language equivalence for finite-state automata, formal language theorists have been studying the equivalence problem over classes of automata which express languages which are more expressive than the class of regular languages generated by finite-state automata. Bar-Hillel, Perles and Shamir [6] were the first to demonstrate, in 1961, that the class of languages defined by context-free grammars was too wide to admit a decidable theory for language equivalence. Shortly after this, Korenjak and Hopcroft [95] demonstrated that language equivalence between simple (deterministic) grammars is decidable. Only recently has the long-open problem of language equivalence between deterministic push-down automata (DPDA) been settled (positively) by Sénizergues [132, 133].

Decidability questions for Petri nets were addressed already two decades ago, with the thesis of Hack [63]. However, it has only been in the much more recent past that a more concerted effort has been focussed on such questions, with the interest driven in part by analogies drawn between classes of concurrent system models and classes of generators for families of formal languages. In [114] Milner

exploits the relationship between regular (finite-state) automata as discussed by Salomaa in [130] and regular behaviours to present the decidability and a complete axiomatisation of bisimulation equivalence for finite-state behaviours, whilst in his textbook [115] he demonstrates that the halting problem for Turing machines can be encoded as a bisimulation question for the full CCS calculus thus demonstrating undecidability in general. This final feat is carried out elegantly using finite representations of counters in the thesis of Taubner [141]. These results are as expected; however, real interest was generated with the discovery by Baeten, Bergstra and Klop [4, 5] that bisimulation equivalence is decidable for a family of infinite-state automata generated by a general class of context-free grammars.

In Section 2, we present an overview of various results obtained regarding decidability and complexity, with particular emphasis on bisimulation equivalence, focussing on the various techniques exploited in each case. Our interest in bisimulation equivalence stems predominantly from its mathematical tractability. Apart from being the fundamental notion of equivalence for several process algebraic formalisms, bisimulation equivalence possesses several pleasing mathematical properties, not least of which being—as we shall discover—that it is decidable over process classes for which all other common equivalences remain undecidable, in particular over the class of processes defined by context-free grammars. Furthermore in a particularly interesting class of processes—namely the normed deterministic processes—all of the standard equivalences coincide, so it is sensible to concentrate on the most mathematically tractable equivalence when analysing properties of another equivalence. In particular, by studying bisimulation equivalence we can rediscover standard theorems about the decidability of language equivalence, as well as provide more efficient algorithms for these decidability results than have previously been presented. We expect that the structural techniques which can be exploited in the study of bisimulation equivalence will prove to be useful in tackling various other language theoretic problems. Indeed, while Sénizergues' proof of the decidability of DPDA is extremely long and complex, developed over 70 pages of a 166 page journal submission, Stirling [138] has since presented a far simpler proof of this result using variations on some of the structural analysis techniques explored in Section 2.

Apart from the basic equivalence checking problem, we also survey related questions regarding ***regularity checking***, that is, determining if a given system is semantically finite-state. Within a typical application domain such as the modelling of protocols, we might analyse infinite-state systems which we intend to semantically represent finite-state behaviours: our specification of the system is finite-state, but the state space of the implementation is (syntactically) infinite. A positive answer for the regularity checking problem would allow the equivalence checking algorithm to exploit well-developed techniques for analysing finite-state systems (assuming that the equivalent finite-state system could be extracted from from this answer).

Model Checking

Apart from equivalence checking, model checking provides perhaps the most promising approach to the formal verification of distributed, reactive systems. In this approach, one uses formulae of a temporal logic to specify the desired properties of a system, and a (semi-)decision procedure then checks whether the given system is a model of the formula at hand. In particular, in the case of finite-state systems model checking is, at least theoretically, always applicable, since an exhaustive traversal through the reachable state space of the system under consideration can effectively provide enough information to solve the verification problem. In fact, using BDDs, finite-state model checking has had some prominent industrial success stories, despite the omni-present state space explosion threat, making it an indispensable tool for hardware design. The practical success of model checking can be witnessed by the immediate popularity of the newly-published textbook by Clarke, Grumberg and Peled [41].

Again, as algorithms for finite-state model checking typically involve the exhaustive traversal of the state space, they are inherently incapable of verifying infinite-state systems, and must be replaced by algorithms based on different techniques. The new techniques which have been developed for model checking have mainly been influenced by formal language theory, where there exists a long tradition of reasoning about finitely-presented infinite objects, in this case formal languages described by automata or grammars. Consequently, notions from formal language theory form a recurring theme in the verification of infinite-state systems, e.g., as a means for the description of such systems, or in characterizing the expressive power of certain temporal logics.

One general problem is, however, the trade-off between expressiveness and decidability. Expressive models of computation, in conjunction with powerful specification formalisms, inevitably lead to an undecidable model checking problem. However, taking weaker models and/or formalisms in order to recover decidability often greatly diminishes its practical value. The goal of this survey is to provide a systematic presentation of the results obtained so far concerning decidability and complexity issues in model checking infinite-state systems, thereby identifying feasible combinations of process classes and temporal logics.

In Section 3.1 we provide a brief introduction to the temporal logics we consider, together with their branching time and linear time classifications. We then proceed by presenting decidability and complexity results about model checking for various classes of infinite-state systems, first for branching time logics in Section 3.2, and then for linear time logics in Section 3.3. Due to the wealth of techniques which have been devised for use in conjunction with the wide variety of logics which one may consider, we only sketch the main ideas of the relevant results in this survey. This gives a conceptual and easy to read overview, which is complemented by tight links to the original literature, thus enabling the interested readers to access all the technical details.

Acknowledgements Sections 1 and 2 are based on the survey *Infinite Results* [118] presented at CONCUR'96, and the origins of Section 3 are found in the survey *More Infinite Results* [25] presented at INFINITY'96. However,

there has been a flood of new results in the intervening years, and these earlier surveys are vastly extended here. We wish to thank Richard Mayr and Markus Mueller-Olm for detailed comments on earlier drafts of this survey.

1 A Taxonomy of Infinite State Processes

1.1 Rewrite Transition Systems

Concurrent systems are modelled semantically in a variety of ways. They may be defined for example by the infinite traces or executions which they may perform, or by the entirety of the properties which they satisfy in some particular process logic, or as a particular algebraic model of some equational specification. In any case, a fundamental unifying view is to interpret such systems as edge-labelled directed graphs, whose nodes represent the states in which a system may exist, and whose transitions represent the possible behaviour of the system originating in the state represented by the node from which the transition emanates; the label on a transition represents an event corresponding to the execution of that transition, which will typically represent an interaction with the environment. The starting point for our study will thus be such graphs, which will for us represent processes.

Definition 1 A *labelled transition system* is a tuple $\langle \mathcal{S}, \Sigma, \longrightarrow, \alpha_0, \mathsf{F} \rangle$ where

- $\mathcal{S}$ is a set of *states*.
- Σ is a finite set of *labels*.
- $\longrightarrow \subseteq \mathcal{S} \times \Sigma \times \mathcal{S}$ is a *transition relation*, written $\alpha \xrightarrow{\mathfrak{a}} \beta$ for $\langle \alpha, \mathfrak{a}, \beta \rangle \in \longrightarrow$.
- $\alpha_0 \in \mathcal{S}$ is a distinguished *start state*.
- $\mathsf{F} \subseteq \mathcal{S}$ is a finite set of *final states* which are *terminal*, meaning that for each $\alpha \in \mathsf{F}$ there is no $\mathfrak{a} \in \Sigma$ and $\beta \in \mathcal{S}$ such that $\alpha \xrightarrow{\mathfrak{a}} \beta$.

This notion of a labelled transition system differs from the standard definition of a finite-state automaton (as for example given in [74]) in that the set of states need not be finite, and final states must not have any outgoing transitions. This last restriction is mild and justified in that a final state refers to the successful termination of a concurrent system. This contrasts with unsuccessful termination (ie, deadlock) which is represented by all non-final terminal states. We could remove this restriction, but only at the expense of Theorem 4 below which characterises a wide class of labelled transition systems as push-down automata which accept on empty stack. (An alternative approach could be taken to recover Theorem 4 based on PDA which accept by final state, but we do not pursue this alternative here.)

We follow the example set by Caucal [32] and consider the families of labelled transition systems defined by various rewrite systems. Such an approach provides us with a clear link between well-studied classes of formal languages and transition system generators, a link which is of particular interest when it comes to exploiting process-theoretic techniques in solving problems in classical formal language theory.

Definition 2 A *sequential labelled rewrite transition system* is a tuple $\langle \mathsf{V}, \Sigma, \mathsf{P}, \alpha_0, \mathsf{F} \rangle$ where

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

A *parallel labelled rewrite transition system* is defined precisely as above, except that the elements of V^* are read modulo commutativity of concatenation, which is thus interpretted as parallel, rather than sequential, composition. We can thus consider states as monomials $X_1^{k_1} X_2^{k_2} \cdots X_n^{k_n}$ over the variables $V = \{X_1, X_2, \cdots, X_n\}$. With this in mind, we shall be able to exploit the following result due to Dickson [48] which is easily proved by induction on n.

Lemma 3 (Dickson's Lemma) *Given an infinite sequence of vectors of natural numbers $\vec{x}_1, \vec{x}_2, \vec{x}_3, \ldots \in \mathbb{N}^n$ we can always find indices i and j with $i < j$ such that $\vec{x}_i \leq \vec{x}_j$ (where $\leq$ is considered pointwise).*

The state $X_1^{k_1} X_2^{k_2} \cdots X_n^{k_n}$ can be viewed as the vector $(k_1, k_2, \ldots, k_n) \in \mathbb{N}^n$. Hence, Dickson's Lemma says that, given any infinite sequence $\alpha_1, \alpha_2, \alpha_3, \ldots$ of such states, we can always find two of these, α_i and α_j with $i < j$, such that the number of occurrences of each variable X in α_j is at least as great as in α_i.

We shall freely extend the transition relation $\longrightarrow$ homomorphically to finite sequences of actions $w \in \Sigma^*$ so as to write $\alpha \xrightarrow{\varepsilon} \alpha$ and $\alpha \xrightarrow{aw} \beta$ whenever $\alpha \xrightarrow{a} \cdot \xrightarrow{w} \beta$. Also, we refer to the set of states α into which the initial state can be rewritten, that is, such that $\alpha_0 \xrightarrow{w} \alpha$ for some $w \in \Sigma^*$, as the *reachable* states. Although we do not insist that all states be reachable, we shall assume that all variables in V are accessible from the initial state, that is, that for all $X \in V$ there is some $w \in \Sigma^*$ and $\alpha, \beta \in V^*$ such that $\alpha_0 \xrightarrow{w} \alpha X \beta$.

This definition is slightly more general than that given in [32], which does not take into account final states nor the possibility of parallel rewriting as an alternative to sequential rewriting. By doing this, we expand the study of the classes of transition systems which are defined, and extend some of the results given by Caucal, notably in the characterisation of arbitrary sequential rewrite systems as push-down automata.

The families of transition systems which can be defined by restricted rewrite systems can be classified using a form of Chomsky hierarchy. 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 in Figure 1. (Type 1 rewrite systems, corresponding to context-sensitive grammars, do not feature in this hierarchy since the rewrite rules by definition are only applied to the prefix of a composition.) In the remainder of this section, we explain the classes of

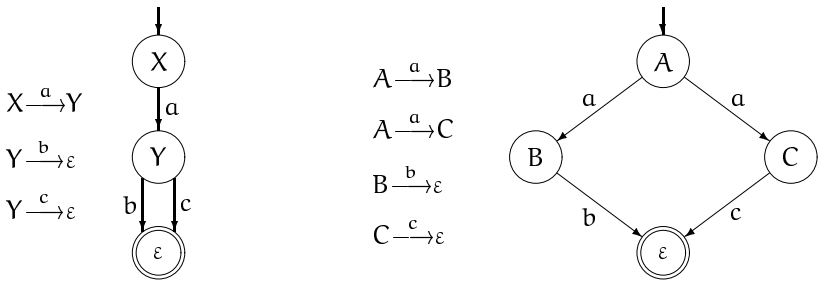
	Restriction on the rules $\alpha \xrightarrow{a} \beta$ of P	Restriction on F	Sequential composition	Parallel composition
Type 0	*none*	*none*	PDA	PN
Type $1\frac{1}{2}$	$\alpha \in Q\Gamma^*$ and $\beta \in Q\Gamma^*$ where $V = Q \uplus \Gamma$	$F = Q$	PDA	MSA
Type 2	$\alpha \in V$	$F = \{\varepsilon\}$	BPA	BPP
Type 3	$\alpha \in V$ and $\beta \in V \cup \{\varepsilon\}$	$F = \{\varepsilon\}$	FSA	FSA

Figure 1: A hierarchy of transition systems.

transition systems which are represented in this table, working upwards starting with the most restrictive class.

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 elements of the finite set $\{\, \alpha \in V^* \,:\, |\alpha| \leq |\alpha_0| \,\}$.

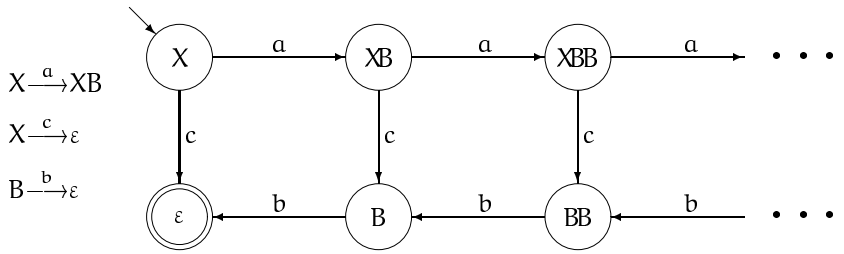
Example 0 *In the following we present two type 3 (regular) rewrite systems along with the FSA transition systems which the initial states* X *and* A*, respectively, denote.*



These two automata both recognise the same (regular) language $\{\, ab, ac \,\}$*. However, they are substantially different automata.*

BPA represents the class of Basic Process Algebra processes of Bergstra and Klop [9], which are the transition systems associated with Greibach normal form (GNF) context-free grammars in which only left-most derivations are permitted.

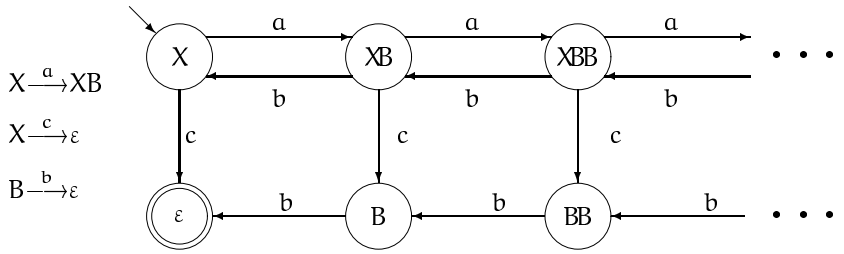
Example 1 *In the following we present a type 2 (GNF context-free grammar) rewrite system along with the BPA transition system which the initial state* X *denotes.*



This automaton recognises the context-free language $\{ a^n c b^n : n \geq 0 \}$.

BPP represents the class of Basic Parallel Processes introduced by Christensen [35] 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.

Example 2 *The type 2 rewrite system from Example 1 gives rise to the following BPP transition system with initial state* X.



This automaton recognises the language consisting of all strings of the form $(a+b)^* c b^*$ *which contain an equal number of* a*'s and* b*'s in which no prefix contains more* b*'s than* a*'s.*

We shall assume that for all type 2 rewrite systems, ε is the only terminal state. Combined with our previous assumption that each variable is reachable from the inital state, this implies that each variables is on the left hand side of at least one rule.

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. Finally, the set of final states is given by Q, which represent the PDA configurations in which the stack is empty.

Caucal [32] demonstrates that, disregarding final states, 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. The stronger result, in which final states are taken into consideration, actually holds as well. The idea behind the encoding is as follows. Given an arbitrary rewrite system $\langle V, \Sigma, P, \alpha_0, F\rangle$, take n satisfying

$$n \geq \max\{\,|\alpha| \,:\, \alpha \xrightarrow{a} \beta \in P\,\}; \qquad \text{and}$$
$$n > \max\{\,|\alpha| \,:\, \alpha \xrightarrow{a} \beta \text{ with } \beta \in F\,\}.$$

Then let

$$Q = \{\,p_\alpha \,:\, \alpha \in V^* \text{ and } |\alpha| < n\,\}; \qquad \text{and}$$
$$\Gamma = \{\,Z_\alpha \,:\, \alpha \in V^* \text{ and } |\alpha| = n\,\} \cup \{\,\Lambda\,\}.$$

Every final transition state $\alpha \in F$ is represented by the PDA state p_α, that is, by the PDA being in control state p_α with an empty stack denoting acceptance; and every non-final transition system state $\gamma\alpha_1\alpha_2\cdots\alpha_k \notin F$ with $|\gamma| < n$ and $|\alpha_i| = n$ for $1 \leq i \leq k$, is represented in the PDA by $p_\gamma Z_{\alpha_1} Z_{\alpha_2} \cdots Z_{\alpha_k}\Lambda$, that is, by the PDA being in control state p_γ with with the sequence $Z_{\alpha_1} Z_{\alpha_2} \cdots Z_{\alpha_k}\Lambda$ on its stack. Then every rewrite rule introduces appropriate PDA rules which mimic it and respect this representation. Thus we arrive at the following result.

Theorem 4 *Every sequential labelled rewrite transition system can be represented (up to the labelling of states) by a PDA transition system.*

Example 3 *The BPP transition system of Example 2 is given by the following sequential rewrite system.*

$$X \xrightarrow{a} XB \qquad X \xrightarrow{c} \varepsilon \qquad B \xrightarrow{b} \varepsilon \qquad XB \xrightarrow{b} X$$

By the above construction, this gives rise to the following PDA with initial state $p_X\Lambda$. (We omit rules corresponding to the unreachable states.)

$\underline{X \xrightarrow{a} XB}$: $p_X\Lambda \xrightarrow{a} p_\varepsilon Z_{XB}\Lambda$ $\quad p_\varepsilon Z_{XB} \xrightarrow{a} p_X Z_{BB}$ $\quad p_X Z_{BB} \xrightarrow{a} p_\varepsilon Z_{XB} Z_{BB}$

$\underline{X \xrightarrow{c} \varepsilon}$: $p_X\Lambda \xrightarrow{c} p_\varepsilon$ $\quad p_\varepsilon Z_{XB} \xrightarrow{c} p_B$ $\quad p_X Z_{BB} \xrightarrow{c} p_\varepsilon Z_{BB}$

$\underline{B \xrightarrow{b} \varepsilon}$: $p_B\Lambda \xrightarrow{b} p_\varepsilon$ $\quad p_\varepsilon Z_{BB} \xrightarrow{b} p_B$ $\quad p_B Z_{BB} \xrightarrow{b} p_\varepsilon Z_{BB}$

$\underline{XB \xrightarrow{b} X}$: $p_\varepsilon Z_{XB} \xrightarrow{b} p_X$ $\quad p_X Z_{BB} \xrightarrow{b} p_\varepsilon Z_{XB}$

This is expressed more simply by the following PDA with initial state $p\Lambda$.

$p\Lambda \xrightarrow{a} pB\Lambda$ $\quad pB \xrightarrow{a} pBB$ $\quad q\Lambda \xrightarrow{b} q$

$p\Lambda \xrightarrow{c} q$ $\quad pB \xrightarrow{b} p$ $\quad qB \xrightarrow{b} q$

$pB \xrightarrow{c} q$

Note that BPA coincides with the class of single-state PDA. However, we shall see in Section 1.3 that any PDA presentation of the transition system of Example 2 must have at least 2 control states: this transition system is not represented by any BPA.

MSA represents the class of multiset automata, which can be viewed as "parallel" or "random-access" push-down automata; they are defined as above except that they have random access capability to the stack.

Example 4 *The BPA transition system of Example 1 is isomorphic to that given by the following MSA with initial state* pX.

$$\mathsf{pX} \xrightarrow{a} \mathsf{pBX} \qquad \mathsf{pX} \xrightarrow{c} \mathsf{q} \qquad \mathsf{qB} \xrightarrow{b} \mathsf{q}$$

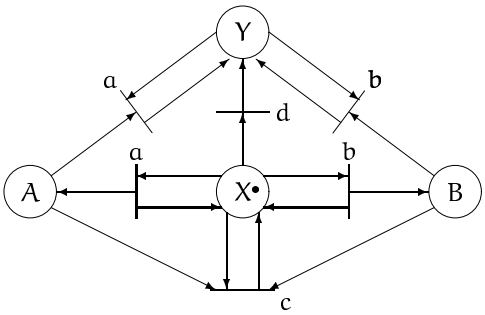
Note that when the stack alphabet has only one element, PDA and MSA trivially coincide. Also note that BPP coincides with the class of single-state MSA. We shall see in Section 1.3 that any MSA presentation of the transition system of Example 1 must have at least 2 control states: this transition system is not represented by any BPP.

PN represents the class of (finite, labellled, 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.

Example 5 *The following unrestricted parallel rewrite system with initial state* X *and final state* Y

$$X \xrightarrow{a} XA \qquad XAB \xrightarrow{c} X \qquad YA \xrightarrow{a} Y$$
$$X \xrightarrow{b} XB \qquad X \xrightarrow{d} Y \qquad YB \xrightarrow{b} Y$$

describes the Petri net which in its usual graphical representation net would be rendered as follows. (The weights on all of the arcs is 1*.)*



The automaton represented by this Petri net recognises the language consisting of all strings from $(a + b + c)^* d(a + b)^*$ *in which the number of* c*'s in any prefix is bounded above by both the number of* a*'s and the number of* b*'s; and in which the number of* a*'s (respectively* b*'s) before the occurrence of the* d *minus the number of* c*'s equals the number of* a*'s (respectively* b*'s) after the occurrence of the* d*.*

Although in the sequential case, PDA constitutes a normal form for unrestricted rewrite transition systems, we shall see in Section 1.3 that this result fails to hold in the parallel case. We shall in fact demonstrate that the automaton associated with the above PN is not the automaton of any MSA.

1.2 Languages and Bisimilarity

Given a labelled transition system $T = \langle \mathcal{S}, \Sigma, \longrightarrow, \alpha_0, F \rangle$, we can define its *language* $L(T)$ to be the language generated by its initial state α_0, where the language generated by a state is defined in the usual fashion as the sequences of actions which label rewrite transitions leading from the given state to a final state.

Definition 5 $L(\alpha) = \{w \in \Sigma^* : \alpha \xrightarrow{w} \beta \text{ for some } \beta \in F\}$, and $L(T) = L(\alpha_0)$. α and β are *language equivalent*, written $\alpha \sim_L \beta$, iff they generate the same language: $L(\alpha) = L(\beta)$.

With respect to the languages generated by rewrite systems, if a rewrite system is in the process of generating a word, then the partial word should be extendible to a complete word. That is, from any reachable state of the transition system, a final state should be reachable. If the transition system satisfies this property, it is said to be *normed*; otherwise it is *unnormed*.

Definition 6 We define the *norm* of any state α of a labelled transition system, written $n(\alpha)$, to be the length of a shortest rewrite transition sequence which takes α to a final state, that is, the length of a shortest word in $L(\alpha)$. By convention, we define $n(\alpha) = \infty$ if there is no sequence of transitions from α to a final state, that is, $L(\alpha) = \emptyset$. The transition system is *normed* iff every reachable state α has a finite norm; otherwise it is *unnormed*.

Note that, due to the assumption following Definition 2 on the accessibility of all the variables, if a type 2 rewrite transition system is normed, then all of its variables must have finite norm. The following then is a basic fact about the norms of type 2 (BPA and BPP) states.

Lemma 7 *Given any state $\alpha\beta$ of a type 2 rewrite transition systems (BPA or BPP),* $n(\alpha\beta) = n(\alpha) + n(\beta)$.

Proof For the sequential case, $L(\alpha\beta) = L(\alpha) \cdot L(\beta) = \{uv : u \in L(\alpha) \text{ and } v \in L(\beta)\}$. For the parallel case, $L(\alpha\beta) = L(\alpha) \parallel L(\beta) = \{u_1v_1u_2v_2 \cdots u_nv_n : u_1u_2 \cdots u_n \in L(\alpha) \text{ and } v_1v_2 \cdots v_n \in L(\beta)\}$. The result follows easily from this. □

A further common property of transition systems is that of *determinacy*.

Definition 8 T is *deterministic* iff for every reachable state α and every label a there is at most one state β such that $\alpha \xrightarrow{a} \beta$.

For example, the two finite state automata presented in Example 0 are both normed transition systems, while only the first is deterministic. All other examples which we have presented have been both normed and deterministic.

In the realm of concurrency theory, language equivalence is generally taken to be too coarse an equivalence. For example, it equates the two transition systems of Example 0 which generate the same language $\{ab, ac\}$ yet demonstrate different deadlocking capabilities due to the nondeterministic behaviour exhibited by the second transition system. Many finer equivalences have been proposed, with *bisimulation equivalence* being perhaps the finest behavioural equivalence studied. (Note that we do not consider here any so-called 'true concurrency' equivalences such as those based on partial orders.) Bisimulation equivalence was defined by Park [125] and used to great effect by Milner [113, 115]. Its definition, in the presence of final states, is as follows.

Definition 9 A binary relation $\mathcal{R}$ on states of a transition system is a *bisimulation* iff whenever $(\alpha, \beta) \in \mathcal{R}$ we have that

- if $\alpha \xrightarrow{a} \alpha'$ then $\beta \xrightarrow{a} \beta'$ for some β' with $(\alpha', \beta') \in \mathcal{R}$;
- if $\beta \xrightarrow{a} \beta'$ then $\alpha \xrightarrow{a} \alpha'$ for some α' with $(\alpha', \beta') \in \mathcal{R}$;
- $\alpha \in F$ iff $\beta \in F$.

α and β are *bisimulation equivalent* or *bisimilar*, written $\alpha \sim \beta$, iff $(\alpha, \beta) \in \mathcal{R}$ for some bisimulation $\mathcal{R}$. That is, $\sim = \bigcup \{ \mathcal{R} : \mathcal{R} \text{ is a bisimulation relation} \}$.

Lemma 10 $\sim$ *is the largest bisimulation relation.*

Proof An arbitrary union of bisimulations is itself a bisimulation. □

Lemma 11 $\sim$ *is an equivalence relation.*

Proof Reflexivity holds since the identity relation is a bisimulation; symmetry holds since the inverse of a bisimulation is a bisimulation; and transitivity holds since the composition of two bisimulations is a bisimulation. □

Bisimulation equivalence has an elegant characterisation in terms of certain two-player games [136, 91]. Starting with a pair of states $\langle \alpha, \beta \rangle$, the two players alternate moves according to the following rules.

1. If exactly one of the pair of states is a final state, then player I is deemed to be the winner. Otherwise, player I chooses one of the states and makes some transition from that state (either $\alpha \stackrel{a}{\longrightarrow} \alpha'$ or $\beta \stackrel{a}{\longrightarrow} \beta'$). If this proves impossible, due to both states being terminal, then player II is deemed to be the winner.

2. Player II must respond to the move made by player I by making an identically-labelled transition from the other state (either $\beta \stackrel{a}{\longrightarrow} \beta'$ or $\alpha \stackrel{a}{\longrightarrow} \alpha'$). If this proves impossible, then player I is deemed to be the winner.

3. The play then repeats itself from the new pair $\langle \alpha', \beta' \rangle$. If the game continues forever, then player II is deemed to be the winner.

The following result is then immediately evident.

Fact 12 $\alpha \sim \beta$ *iff Player II has a winning strategy in the bisimulation game starting with the pair* $\langle \alpha, \beta \rangle$.

Conversely, $\alpha \not\sim \beta$ *iff Player I has a winning strategy in the bisimulation game starting with the pair* $\langle \alpha, \beta \rangle$.

Proof Any bisimulation relation defines a winning strategy for player II for the bisimulation game starting from a pair in the relation: the second player merely has to respond to moves by the first in such a way that the resulting pair is contained in the bisimulation.

Conversely, a winning strategy for player II for the bisimulation game starting from a particular pair of states defines a bisimulation relation containing that pair, namely the collection of all pairs which appear after every exchange of moves during any and all games in which player II uses this strategy. □

Example 6 *The states* X *and* A *from Example 0 are language equivalent, as they define the same language* $\{\,\mathtt{ab}, \mathtt{ac}\,\}$. *However, they are not bisimulation equivalent: there is no bisimulation relation which relates them. To see this, we can demonstrate an obvious winning strategy for player I in the bisimulation game starting with the pair of states* $\langle \mathsf{X}, \mathsf{A}\rangle$. *After the first exchange of moves, the new pair of states must be either* $\langle \mathsf{Y}, \mathsf{B}\rangle$ *or* $\langle \mathsf{Y}, \mathsf{C}\rangle$*; in the former case, player I may make the transition* $\mathsf{Y} \xrightarrow{\mathtt{c}} \varepsilon$ *to which player I cannot respond from* B, *and in the latter case player I may make the transition* $\mathsf{Y} \xrightarrow{\mathtt{b}} \varepsilon$ *to which player I cannot respond from* C.

A transition system is *finite-branching* if there are only a finite number of transitions from any given reachable state; and it is *image-finite* if there are only a finite number of transitions with a given label from any given reachable state. For image-finite transition systems, we have the following stratified characterisation of bisimulation equivalence [115].

Definition 13 The *stratified bisimulation relations* $\sim_n$ are defined as follows.

- $\alpha \sim_0 \beta$ for all states.
- $\alpha \sim_{k+1} \beta$ iff
  - if $\alpha \xrightarrow{a} \alpha'$ then $\beta \xrightarrow{a} \beta'$ for some β' with $\alpha' \sim_k \beta'$;
  - if $\beta \xrightarrow{a} \beta'$ then $\alpha \xrightarrow{a} \alpha'$ for some α' with $\alpha' \sim_k \beta'$;
  - $\alpha \in \mathsf{F}$ iff $\beta \in \mathsf{F}$.

In terms of the game characterisation of bisimilarity, $\alpha \sim_n \beta$ iff Player I cannot force a win within the first n exchanges of moves.

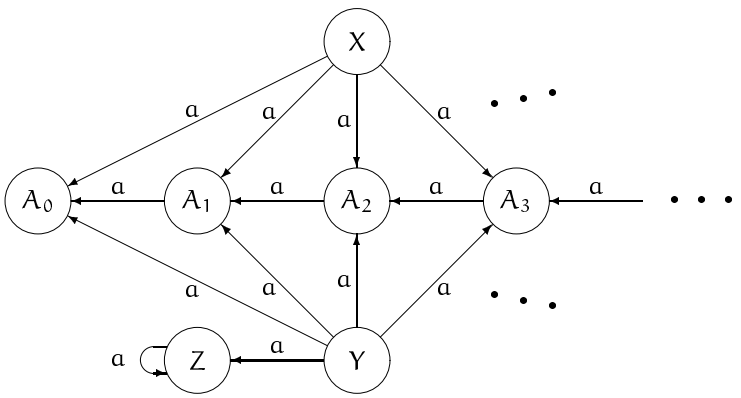
Lemma 14 *If* α *and* β *are image-finite, then* $\alpha \sim \beta$ *iff* $\alpha \sim_n \beta$ *for all* $n \geq 0$.

Proof If $\alpha \sim \beta$ then an induction proof gives that $\alpha \sim_n \beta$ for each $n \geq 0$.

Conversely, $\{(\alpha, \beta) : \alpha$ and β are image-finite and $\alpha \sim_n \beta$ for all $n \geq 0\}$ is a bisimulation. □

It is clear that rewrite transition systems are image-finite, and that this lemma applies. It is equally clear that each of the relations $\sim_n$ is decidable, and that therefore non-bisimilarity is semi-decidable over rewrite transition systems. Hence the decidability results would follow from demonstrating the semi-decidability of bisimilarity.

Example 7 *Consider the following infinite-branching transition system.*



The states X *and* Y *are clearly not bisimilar, as the state* Z *cannot be bisimilar to* A_n *for any* $n \geq 0$. *However,* $X \sim_n Y$ *for every* $n \geq 0$ *as* $Z \sim_n A_n$ *for each* $n \geq 0$.

An immediately evident yet important property is the following lemma with its accompanying corollary relating bisimulation equivalence to language equivalence.

Lemma 15 *If* $\alpha \sim \beta$ *and* $\alpha \stackrel{w}{\longrightarrow} \alpha'$ *with* $w \in \Sigma^*$, *then* $\beta \stackrel{w}{\longrightarrow} \beta'$ *for some* β' *such that* $\alpha' \sim \beta'$.

Proof Given a bisimulation $\mathcal{R}$ containing (α, β), an induction on the length of $w \in \Sigma^*$ demonstrates that if $\alpha \stackrel{w}{\longrightarrow} \alpha'$ then $\beta \stackrel{w}{\longrightarrow} \beta'$ with $(\alpha', \beta') \in \mathcal{R}$. □

Corollary 16 *If* $\alpha \sim \beta$ *then* $\alpha \sim_L \beta$.

Proof If $\alpha \sim \beta$ then $w \in L(\alpha)$ iff $\alpha \stackrel{w}{\longrightarrow} \alpha'$ where α' is a final state, which—by the previous lemma—holds iff $\beta \stackrel{w}{\longrightarrow} \beta'$ where β' is a final state, which in turn holds iff $w \in L(\beta)$. □

Apart from being the fundamental notion of equivalence for several process algebraic formalisms, bisimilarity has several pleasing mathematical properties not shared by other equivalences such as language equivalence. Furthermore as given by the following lemma, language equivalence and bisimilarity coincide over the class of normed deterministic processes.

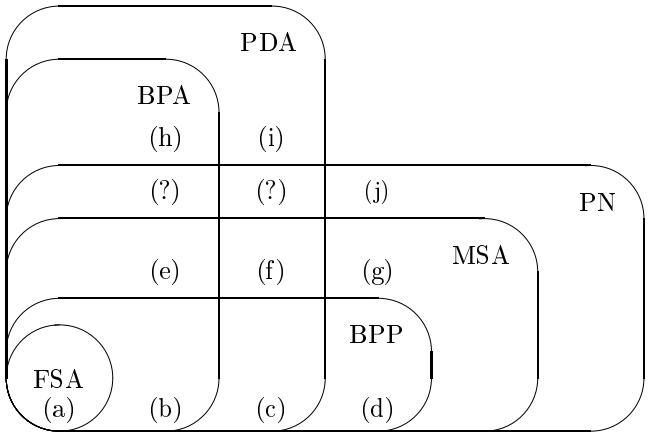
Lemma 17 *For states* α *and* β *of a normed deterministic transition system, if* $\alpha \sim_L \beta$ *then* $\alpha \sim \beta$. *Thus, taken along with Corollary 16,* $\sim_L$ *and* $\sim$ *coincide.*

Proof It suffices to demonstrate that the relation $\{(\alpha, \beta) : \alpha \sim_L \beta$ and α, β are states of a normed deterministic transition system$\}$ is a bisimulation relation. □

Hence it is sensible to concentrate on the more mathematically tractable bisimulation equivalence when investigating decidability results for language equivalence for deterministic language generators. We shall see examples of this in Section 2.

1.3 Expressivity Results

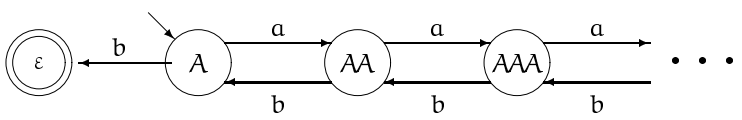
The hierarchy from above gives us the following classification of processes.



In this section we demonstrate the strictness of this hierarchy by providing example transition systems which lie precisely in the gaps (a)-(j) indicated in the classification. (We leave open the question regarding the final two gaps.) We in fact do more than this by giving examples of normed deterministic transition systems which separate all of these classes up to bisimulation (and hence also language) equivalence. These results complement those presented for the taxonomy described by Burkart, Caucal and Steffen [24].

(a) The first automaton in example 0 provides a normed deterministic FSA.

(b) The type 2 rewrite system with the two rules $A \xrightarrow{a} AA$ and $A \xrightarrow{b} \varepsilon$ gives rise to the same transition system regardless of whether the system is sequential or parallel; this is an immediate consequence of the fact that it involves only a single variable A. This transition system is depicted as follows.



This is an example of a normed deterministic transition system which is both a BPA and a BPP but not an FSA.

(c) Examples 2 and 3 provide a transition system which can be described by both a BPP (Example 2) and a PDA (Example 3). However, it cannot be described up to bisimilarity by any BPA. To see this, suppose that

we have a BPA which represents this transition system up to bisimilarity, and let m be at least as large as the norm of any of its variables. Then the BPA state corresponding to XB^m in Example 2 must be of the form $A\alpha$ where $A \in V$ and $\alpha \in V^+$. But then *any* sequence of $n(A)$ norm-reducing transitions must lead to the BPA state α, while the transition system in Example 2 has two such non-bisimilar derived states, namely XB^{k-1} and B^k where $k = n(\alpha)$.

(d) The following BPP with initial state X

$$X \xrightarrow{a} XB \qquad X \xrightarrow{c} XD \qquad X \xrightarrow{e} \varepsilon \qquad B \xrightarrow{b} \varepsilon \qquad D \xrightarrow{d} \varepsilon$$

is not language equivalent to any PDA, as its language is easily confirmed not to be context free. (The words in this language of the form $a^*c^*b^*d^*e$ are exactly those of the form $a^kc^nb^kd^ne$, which is clearly not a context-free language.)

(e) Examples 1 and 4 provide a transition system which can be described by both a BPA (Example 1) and a MSA (Example 4). However, the context-free language which it generates, $\{a^ncb^n : n \geq 0\}$, cannot be generated by any BPP, so this transition system is not even language equivalent to any BPP. To see this, suppose that $L(X) = \{a^ncb^n : n \geq 0\}$ for some BPP state X. (As the process has norm 1, the state must consist of a single variable X.) Let k be at least as large as the norm of any of the finite-normed variables of this BPP, and consider a transition sequence accepting the word a^kcb^k:

$$X \xrightarrow{a^k} Y\alpha \xrightarrow{c} \beta\alpha \xrightarrow{b^k} \varepsilon$$

where the c-transition is generated by the transition rule $Y \xrightarrow{c} \beta$. We must have $n(Y\alpha) = k+1 > n(Y)$, so $\alpha \neq \varepsilon$; hence $\alpha \xrightarrow{b^i} \varepsilon$ and $\beta \xrightarrow{b^{k-i}} \varepsilon$ for some $i > 0$. Thus we have

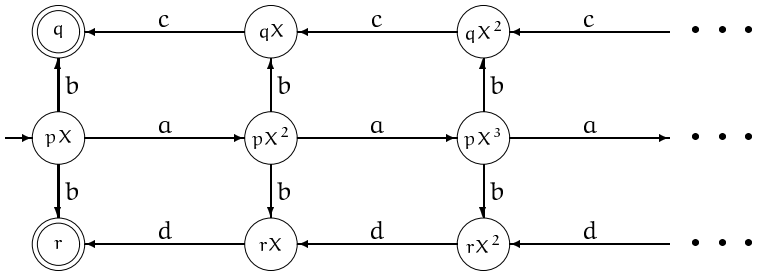
$$X \xrightarrow{a^k} Y\alpha \xrightarrow{b^i} Y \xrightarrow{c} \beta \xrightarrow{b^{k-i}} \varepsilon$$

from which we get the desired contradiction: $a^kb^icb^{k-i} \in L(X)$ for some $i > 0$.

(f) The following PDA with initial state pX

$$pX \xrightarrow{a} pXX \qquad pX \xrightarrow{b} q \qquad pX \xrightarrow{b} r \qquad qX \xrightarrow{c} q \qquad rX \xrightarrow{d} r$$

coincides with the MSA which it defines, since there is only one stack symbol. This transition system is depicted as follows.



However, this transition system cannot be bisimilar to any BPA, due to a similar argument as for (c), nor language equivalent to any BPP, due to a similar argument as for (e).

(g) The following MSA with initial state pX

$$pX \xrightarrow{a} pA \qquad pA \xrightarrow{a} pAA \qquad qA \xrightarrow{b} qB \qquad rA \xrightarrow{c} r$$
$$pA \xrightarrow{b} qB \qquad qB \xrightarrow{c} r \qquad rB \xrightarrow{c} r$$

generates the language $\{a^n b^k c^n \ : \ 0 < k \leq n\}$, and hence cannot be language equivalent to any PDA, as it is not a context-free language, nor to any BPP, due to a similar argument as for (e).

(h) The following BPA with initial state X

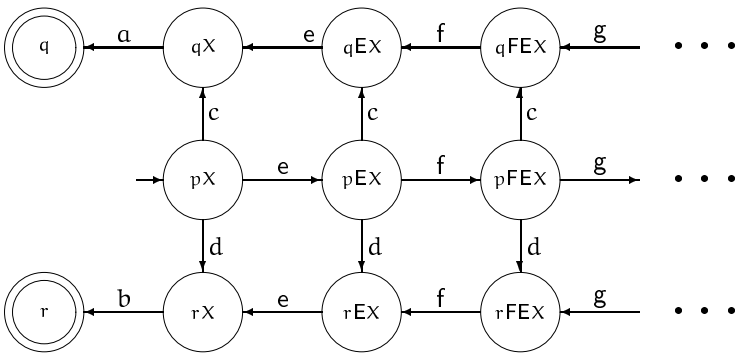
$$X \xrightarrow{a} XA \qquad X \xrightarrow{b} XB \qquad X \xrightarrow{c} \varepsilon \qquad A \xrightarrow{a} \varepsilon \qquad B \xrightarrow{b} \varepsilon$$

generates the language $\{wcw^R \ : \ w \in \{a, b\}^*\}$ and hence is not language equivalent to any PN [126].

(i) The following PDA with initial state pX

$$pX \xrightarrow{a} pAX \quad pA \xrightarrow{a} pAA \quad pB \xrightarrow{a} pAB \quad qA \xrightarrow{a} q \quad rA \xrightarrow{a} r$$
$$pX \xrightarrow{b} pBX \quad pA \xrightarrow{b} pBA \quad pB \xrightarrow{b} pBB \quad qB \xrightarrow{b} q \quad rB \xrightarrow{b} r$$
$$pX \xrightarrow{c} qX \quad pA \xrightarrow{c} qA \quad pB \xrightarrow{c} qB \quad qX \xrightarrow{a} q \quad rX \xrightarrow{b} r$$
$$pX \xrightarrow{d} rX \quad pA \xrightarrow{d} rA \quad pB \xrightarrow{d} rB$$

is constructed by combining the ideas from (f) and (h). It can be schematically pictured as follows.



In this picture, $\mathsf{e}, \mathsf{f}, \mathsf{g}, \ldots \in \{\mathsf{a}, \mathsf{b}\}$ and $\mathsf{E}, \mathsf{F}, \mathsf{G}, \ldots \in \{\mathsf{A}, \mathsf{B}\}$ correspond in the obvious way. The language this PDA generates is $\{ w\mathsf{c}w^{\mathrm{R}}\mathsf{a},\ w\mathsf{c}w^{\mathrm{R}}\mathsf{b} : w \in \{\mathsf{a}, \mathsf{b}\}^* \}$ and hence as in (h) above it is not language equivalent to any PN; and as in (c) above it is not bisimilar to any BPA.

(j) The Petri net from Example 5 cannot be language equivalent to any PDA, as its language is easily confirmed not to be context-free. (The words in this language of the form $\mathsf{a}^*\mathsf{b}^*\mathsf{c}^*\mathsf{d}$ are exactly those of the form $\mathsf{a}^n\mathsf{b}^n\mathsf{c}^n\mathsf{d}$, which is clearly not a context-free language.)

More importantly, this Petri net cannot be bisimilar to any MSA [119]. To see this, suppose that the net is bisimilar to some MSA state $\mathsf{p}A$. (As the process has norm 1, the stack must consist of a single symbol A.) Consider performing an indefinite sequence of a-transitions from $\mathsf{p}A$. By Dickson's Lemma (Lemma 3), we must eventually pass through two states $\mathsf{q}\alpha$ and $\mathsf{q}\alpha\beta$ in which the control states are equal and the stack of the first is contained in the stack of the second. This implies is that we can perform the following execution sequence.

$$\mathsf{p}A \xrightarrow{\mathsf{a}^k} \mathsf{q}\alpha \xrightarrow{\mathsf{a}^k} \mathsf{q}\alpha\beta \xrightarrow{\mathsf{a}^k} \mathsf{q}\alpha\beta^2 \xrightarrow{\mathsf{a}^k} \cdots$$

(We can assume that the period of the cycle is of the same length as the initial segment. If this isn't already given by the Lemma, then we can merely extend the initial segment to the next multiple of the length of the cycle given by the Lemma, and use this multiple as the cycle length.) Considering now an indefinite sequence of b-transitions from $\mathsf{q}\alpha$, a second application of Dickson's Lemma gives us the following execution sequence.

$$\mathsf{q}\alpha \xrightarrow{\mathsf{b}^k} \mathsf{r}\gamma \xrightarrow{\mathsf{b}^k} \mathsf{r}\gamma\delta \xrightarrow{\mathsf{b}^k} \mathsf{r}\gamma\delta^2 \xrightarrow{\mathsf{b}^k} \cdots$$

(We can assume again by the same reasoning as above that the period of the cycle is of the same length as the initial sequence. Furthermore, we can assume that this is the same as the cycle length of the earlier a-sequence, by redefining the cycle lengths to be a common multiple of the two cycle lengths provided by the Lemma.) Now there must be a state $\mathsf{s}\sigma$ such that

$$pA \xrightarrow{a^k} q\alpha \xrightarrow{b^k} r\gamma \xrightarrow{c^k} s\sigma \not\xrightarrow{c}.$$

Consider then the following sequence of transitions.

$$pA \xrightarrow{a^{2k}} q\alpha\beta \xrightarrow{b^{2k}} r\gamma\delta\beta \xrightarrow{c^k} s\sigma\delta\beta \xrightarrow{c}$$

There must be a rule for $sX\xrightarrow{c}$ for some X which appears in either δ or β. But considering the following sequence of transitions

$$pA \xrightarrow{a^k} q\alpha \xrightarrow{b^{2k}} r\gamma\delta \xrightarrow{c^k} s\sigma\delta \not\xrightarrow{c}$$

we must deduce that this X cannot appear in δ. Equally, considering the following sequence of transitions

$$pA \xrightarrow{a^{2k}} q\alpha\beta \xrightarrow{b^k} r\gamma\beta \xrightarrow{c^k} s\sigma\beta \not\xrightarrow{c}$$

we must deduce that this X cannot appear in β. We thus have the desired contradiction.

We here summarize again these separation results in the following theorem.

Theorem 18 *There exist (normed and deterministic) labelled transition systems lying precisely in the gaps* (a)–(j) *in the figure above.*

1.4 Further Classes of Processes

For concreteness, in this chapter we concentrate mainly on the classes of processes which fall within the described hierarchy. However, we shall also summarise various decidability and algorithmic results pertaining to other classes of processes which have been intensively studied. These are either extensions or restrictions of classes within our hierarchy, and are summarised below.

PA Historically, the class BPA was first defined as a process algebra [9] with prefixing, nondeterministic choice, sequential composition, and recursion. Similarly, the class BPP was introduced by Christensen [35] as the parallel analogue to BPA, where sequential composition is replaced by parallel composition without communication. A natural extension of both classes then is their least common generalization, obtained by allowing both kinds of composition simultaneously. This class of processes is known simply as *Process Algebra* (PA).

One-counter automata If we restrict the stack alphabet of a PDA so that it contains only one symbol (apart from a special bottom-of-stack marker), then we arrive at the class of *one-counter automata*. This is the class defined by finite-state automata augmented with a single counter variable; transitions may increment, decrement, or leave unchanged the value of the counter, and such a transition may be determined by a test-for-zero of the counter value. Note that with two counters we describe the full power of Turing machines, and hence we cannot hope for decidability for any semantic notions. Restricting to a single counter aims to delimit the decidability boundary.

One-counter nets The class of Petri nets with (at most) one unbounded place gives rise to the class of *one-counter nets*. These differ from one-counter automata in that they cannot test-for-zero, but rather only test-for-nonzero. That is, if a transition may occur if the counter is zero, then it can equally occur if the counter is non-zero. As we shall see, various problems will be undecidable for Petri nets with only two unbounded places; this being the case, considering Petri nets with only a single unbounded place makes for a natural restriction.

2 The Equivalence Checking Problem

2.1 Decidability Results for BPA and BPP

In this section we describe several positive results which have been established regarding the decidability of bisimilarity between type 2 rewrite transition systems. In particular, we shall briefly describe the techniques behind the following results:

1. Bisimilarity is decidable for BPA [38] and BPP [37, 36].
2. Bisimilarity is decidable in polynomial time for normed BPA [69, 70] and normed BPP [71].

These results contrast with those regarding the undecidability of language equivalence for both BPA and BPP. The negative result for BPA [6] follows from the fact that BPA effectively defines the class of context-free languages; and that for BPP [66] follows from a modification by Hirshfeld of a technique of Jančar which is described in Section 2.4. Both arguments can be shown to hold for the class of normed systems. Also, for both BPA and BPP, this undecidability extends to all equivalences which lie in Glabbeek's spectrum [59] between bisimilarity and language equivalence [82, 61, 79].

Baeten, Bergstra and Klop [4, 5] presented the first such decidability result, that bisimilarity between normed BPA is decidable. Their lengthy proof exploits the periodicity which exists in normed BPA transition systems, and several simpler proofs exploiting structural properties were soon recorded, notably by Caucal [31], Hüttel and Stirling [80], and Groote [60]. Huynh and Tian [81] demonstrated that this problem has a complexity of Σ_2^{P} by providing a nondeterministic algorithm which relies on an NP oracle; Hirshfeld, Jerrum and Moller [69, 70] refined this result by providing a polynomial algorithm, thus showing the problem to be in P. A generally more efficient, though worst-case exponential, algorithm is presented by Hirshfeld and Moller [72]. Finally, Christensen, Hüttel and Stirling [38, 39] demonstrated that bisimilarity between arbitrary BPA systems is decidable, whilst Burkart, Caucal and Steffen [23] demonstrated an elementary decision procedure. For the parallel case, Christensen, Hirshfeld and Moller [37, 36] demonstrated the general decidability result for BPP, whilst Hirshfeld, Jerrum and Moller [71] presented a polynomial algorithm for the normed case.

As we noted in Section 1.2, one direction for the decidability results, determining non-bisimilarity, is automatic using the stratified bisimulation relations of Definition 13. It therefore behoves us merely to prove that bisimilarity is semi-decidable.

2.1.1 Composition and Decomposition

An important property of bisimulation equivalence which we shall exploit is the following congruence result, which is valid for both BPA and BPP.

Lemma 19 *$\sim$ is a congruence (with respect to concatenation) over both BPA and BPP; that is, if $\alpha \sim \beta$ and $\alpha' \sim \beta'$ then $\alpha\alpha' \sim \beta\beta'$.*

Proof $\left\{ (\alpha\alpha', \beta\beta') \ : \ \alpha \sim \beta \text{ and } \alpha' \sim \beta' \right\}$ is a bisimulation. □

Given this congruence property, we have an obvious potential technique for the analysis of (a state of) a type 2 rewrite transition system: decompose the state into simpler components. In general, this idea is not suitably provided for by this congruence property; states may indeed decompose, but not necessarily into simpler components, nor necessarily in any unique fashion. However, for the class of normed transition systems we have a unique factorisation result for both BPA and BPP, of the form first explored by Milner and Moller [116]. Given a type 2 rewrite transition system, we say that a variable $X \in V$ is *prime* (with respect to bisimilarity $\sim$) iff $\alpha = \varepsilon$ whenever $X \sim Y\alpha$; and both normed BPA states and normed BPP states can be decomposed in a unique fashion into such prime components. We shall demonstrate these two results here separately. For the BPA case, we start with the following cancellation lemma.

Lemma 20 (Cancellation Lemma for normed BPA) *If α, β and γ are states of a BPA and if $n(\gamma) < \infty$, then $\alpha\gamma \sim \beta\gamma$ implies $\alpha \sim \beta$.*

Proof $\left\{ (\alpha, \beta) \ : \ \alpha\gamma \sim \beta\gamma \text{ for some } \gamma \text{ with } n(\gamma) < \infty \right\}$ is a bisimulation. □

The assumption in this lemma that γ is normed cannot be dropped, as can be readily seen by considering the following counterexample. Given the two rewrite rules $X \xrightarrow{a} \varepsilon$ and $Y \xrightarrow{a} Y$, we can immediately deduce that $XY \sim Y$, but that $X \not\sim \varepsilon$.

Theorem 21 (Unique Factorization Theorem for normed BPA) *Every normed state α of a BPA decomposes uniquely (up to bisimilarity) into prime components.*

Proof The existence of a prime decomposition may be established by induction on the norm.

For uniqueness, suppose that $\alpha = X_1 \ldots X_p \sim Y_1 \ldots Y_q$ are prime decompositions, and that we have established the uniqueness of prime decompositions for all states β with $n(\beta) < n(\alpha)$. If $p = 1$ or $q = 1$ then uniqueness is immediate. Otherwise suppose that $X_1 \xrightarrow{a} \gamma$ is a norm-reducing transition that is matched by $Y_1 \xrightarrow{a} \delta$, so that $\gamma X_2 \ldots X_p \sim \delta Y_2 \ldots Y_q$. By the inductive hypothesis, the prime decompositions of these two states are equal (up to $\sim$), entailing $X_p \sim Y_q$. Hence, by Lemmas 19 and 20, $X_1 \ldots X_{p-1} \sim Y_1 \ldots Y_{q-1}$, and uniqueness then follows from a second application of the inductive hypothesis. □

Notice that this theorem fails for unnormed BPA states, a fact which is immediately apparent from the observation that $\alpha \sim \alpha\beta$ whenever $n(\alpha) = \infty$.

Theorem 22 (Unique Factorization Theorem for normed BPP) *Every normed state of a BPP decomposes uniquely (up to bisimilarity) into prime components.*

Proof Again, the existence of a prime decomposition may be established by induction on the norm.

For uniqueness, suppose that $\alpha = P_1^{k_1} P_2^{k_2} \cdots P_m^{k_m} \sim P_1^{l_1} P_2^{l_2} \cdots P_m^{l_m} = \beta$ represents a counterexample of smallest norm; that is, all states γ with $n(\gamma) < n(\alpha)$ have unique prime decompositions, the P_is are distinct (with respect to $\sim$) primes, but there is some i such that $k_i \neq l_i$. We may assume that the P_is are ordered by nondecreasing norms, and then we may choose this i so that $k_j = l_j$ whenever $j > i$. We shall furthermore assume without loss of generality that $k_i > l_i$. We distinguish three cases, and in each case show that the state α may perform a norm-reducing transition $\alpha \xrightarrow{a} \alpha'$ that cannot be matched by any transition $\beta \xrightarrow{a} \beta'$ with $\alpha' \sim \beta'$ (or vice versa with the roles of α and β reversed), which will supply the desired contradiction. Observe that by minimality of the counterexample, if α' and β' are to be bisimilar then their prime decompositions must be identical.

Case I. If $k_j > 0$ for some $j < i$, then we may let α perform some norm-reducing transition via the prime component P_j. β cannot match this transition, as it cannot increase the exponent l_i without decreasing the exponent of some prime with norm greater than that of P_i.

Case II. If $k_j > 0$ for some $j > i$, then we may let α perform a norm-reducing transition via the prime component P_j that maximises (after reduction into primes) the increase in the exponent k_i. Again β is unable to match this transition.

Case III. If the state $\alpha = P_i^{k_i}$ is a prime power, then note that $l_j = 0$ for all $j > i$ by choice of i, and that $k_i \geq 2$ by the definition of "prime." If $l_i > 0$, then we may let β perform a norm-reducing transition via P_i; this transition cannot be matched by α, since it would require the exponent k_i to decrease by at least two. If $l_i = 0$ on the other hand, then we may let α perform a norm-reducing transition via P_i; this transition cannot be matched by β, since β is unable to increase the exponent l_i.

These cases are inclusive, so the theorem is proved. □

The following then is an immediate corollary.

Lemma 23 (Cancellation Lemma for normed BPP) *If α, β and γ are normed states of a BPP, then $\alpha\gamma \sim \beta\gamma$ implies $\alpha \sim \beta$.*

Notice that for unnormed BPP, unique decomposition and cancellation again each fail, for similar reasons to the sequential case.

Given a binary relation $\mathcal{R}$ on states of a type 2 rewrite transition system, let $\overset{\mathcal{R}}{\equiv}$ denote the least congruence containing $\mathcal{R}$; that is, $\overset{\mathcal{R}}{\equiv}$ is the least equivalence relation which contains $\mathcal{R}$ and the pair $(\alpha\alpha', \beta\beta')$ whenever it contains each of (α, β) and (α', β').

Definition 24 A binary relation $\mathcal{R}$ on states of a type 2 rewrite transition system is a *bisimulation base* iff whenever $(\alpha, \beta) \in \mathcal{R}$ we have that

- if $\alpha \stackrel{a}{\longrightarrow} \alpha'$ then $\beta \stackrel{a}{\longrightarrow} \beta'$ for some β' with $\alpha' \stackrel{\mathcal{R}}{\equiv} \beta'$;
- if $\beta \stackrel{a}{\longrightarrow} \beta'$ then $\alpha \stackrel{a}{\longrightarrow} \alpha'$ for some α' with $\alpha' \stackrel{\mathcal{R}}{\equiv} \beta'$.

Hence the definition of a bisimulation base differs from that of a bisimulation only in how the derivative states α' and β' are related; in defining $\mathcal{R}$ to be a bisimulation, we would need these derivative states to be related by $\mathcal{R}$ itself and not just by the (typically much larger) congruence $\stackrel{\mathcal{R}}{\equiv}$. A bisimulation base then is in some sense a basis for a bisimulation. The importance of this idea is encompassed in the following theorem due originally (in the normed BPA case) to Caucal.

Lemma 25 *If $\mathcal{R}$ is a bisimulation base for a BPA or BPP transition system, then $\stackrel{\mathcal{R}}{\equiv}$ is a bisimulation, and hence $\stackrel{\mathcal{R}}{\equiv} \subseteq \sim$.*

Proof If $\alpha \stackrel{\mathcal{R}}{\equiv} \beta$ then the two clauses given by the definition of $\stackrel{\mathcal{R}}{\equiv}$ being a bisimulation hold true. This can be demonstrated by a straightforward induction on the depth of inference of $\alpha \stackrel{\mathcal{R}}{\equiv} \beta$. □

Corollary 26 $\alpha \sim \beta$ *iff* $(\alpha, \beta) \in \mathcal{R}$ *for some bisimulation base* $\mathcal{R}$.

Proof Immediate from Lemma 25. □

It now becomes apparent that in order to demonstrate bisimilarity between terms, we needn't produce a complete (infinite) bisimulation relation which contains the pair; rather it suffices simply to produce a bisimulation base which contains the pair. What we shall demonstrate is that this corollary can be strengthened to a finite characterisation of bisimulation, in that we shall prove the existence for both BPA and BPP of a *finite* relation $\mathcal{R}$ satisfying $\stackrel{\mathcal{R}}{\equiv} = \sim$. These relations $\mathcal{R}$ will clearly be bisimulation bases, and the semi-decidability results (and hence the decidability results) will be established, taking into account the following.

Lemma 27 *It is semi-decidable whether a given finite binary relation $\mathcal{R}$ over V^* is a bisimulation base.*

Proof We need simply check that each pair (α, β) of the finite relation $\mathcal{R}$ satisfies the two clauses of the definition of a bisimulation base, which requires testing (in parallel) if each transition for one of α and β has a matching transition from the other. This matching test—that is, checking if the derivative states are related by $\stackrel{\mathcal{R}}{\equiv}$—is itself semi-decidable, as the relation $\stackrel{\mathcal{R}}{\equiv}$ is semi-decidable. □

The semi-decision procedure for checking $\alpha \sim \beta$ would then consists of enumerating all finite binary relations over V^* containing the pair (α, β) and checking (in parallel) if any one of them is a bisimulation base. We thus concentrate on defining the finite relation $\mathcal{R}$ satisfying $\stackrel{\mathcal{R}}{\equiv} = \sim$.

2.1.2 A Finite Bisimulation Base for BPA

The finite bisimulation base for BPA which we present here is taken from Christensen, Hüttel and Stirling [38, 39]. We first present some technical results, starting with the following unique solutions lemma.

Lemma 28 (Unique Solutions Lemma) *If* $\alpha \sim \gamma\alpha$ *and* $\beta \sim \gamma\beta$ *for some* $\gamma \neq \varepsilon$ *then* $\alpha \sim \beta$.

Proof Let $\mathcal{R} = \left\{ (\delta\alpha, \delta\beta) \ : \ \alpha \sim \gamma\alpha \text{ and } \beta \sim \gamma\beta \text{ for some } \gamma \neq \varepsilon \right\}$. We may demonstrate straightforwardly that $\sim\mathcal{R}\sim$ is a bisimulation, from which we may deduce the desired result. □

An important finiteness result on which the construction of the finite bisimulation base hangs is given by the following.

Lemma 29 *If* $\alpha\gamma \sim \beta\gamma$ *for infinitely many non-bisimilar* γ, *then* $\alpha \sim \beta$.

Proof We can show that $\mathcal{R} = \left\{ (\alpha, \beta) \ : \ \alpha\gamma \sim \beta\gamma \text{ for infinitely many non-bisimilar } \gamma \right\}$ is a bisimulation, from which the result follows. □

We may split the set of variables into two disjoint sets $\mathsf{V} = \mathsf{V}_{\mathsf{N}} \cup \mathsf{V}_{\mathsf{U}}$ with the variables in V_{N} having finite norm and those in V_{U} having infinite norm. The motive in this is based on the following lemma.

Lemma 30 *If* X *has infinite norm then* $\mathsf{X}\alpha \sim \mathsf{X}$ *for all* α.

Proof We can immediately verify that $\left\{ (\mathsf{X}, \mathsf{X}\alpha) \ : \ \mathsf{X} \text{ has infinite norm} \right\}$ is a bisimulation. □

Hence we need only ever consider states $\alpha \in \mathsf{V}_{\mathsf{N}}^* \cup \mathsf{V}_{\mathsf{N}}^*\mathsf{V}_{\mathsf{U}}$, the others being immediately transformed into such a bisimilar state by erasing all symbols following the first infinite-norm variable.

The construction relies on recognising when a term may be broken down into a composition of somehow simpler terms. To formalise this concept we start with the following definition.

Definition 31 *A pair* $(\mathsf{X}\alpha, \mathsf{Y}\beta)$ *satisfying* $\mathsf{X}\alpha \sim \mathsf{Y}\beta$ *is* decomposable *if* X *and* Y *have finite norm, and for some* γ,

- $\mathsf{X} \sim \mathsf{Y}\gamma$ *and* $\gamma\alpha \sim \beta$; *or*
- $\mathsf{Y} \sim \mathsf{X}\gamma$ *and* $\gamma\beta \sim \alpha$.

The situation would be clear if all bisimilar pairs were decomposable; indeed we shall exploit this very property of normed transition systems—which follows there from the unique decomposability result—in Section 2.2.1. However we can demonstrate that there is in some sense only a finite number of ways that decomposability can fail. This crucial point in the argument is formalised in the following lemma. We consider two pairs $(\mathsf{X}\alpha, \mathsf{Y}\beta)$ and $(\mathsf{X}\alpha', \mathsf{Y}\beta')$ to be *distinct* if $\alpha \not\sim \alpha'$ or $\beta \not\sim \beta'$.

Lemma 32 *For any* $X, Y \in V$*, any set* $\mathcal{R}$ *of the form*

$$\left\{ (X\alpha, Y\beta) \,:\, X\alpha, Y\beta \in V_N^* \cup V_N^* V_U,\ X\alpha \sim Y\beta, \text{ and } (X\alpha, Y\beta) \text{ not decomposable} \right\}$$

which contains only distinct pairs must be finite.

Proof If $X, Y \in V_U$ then clearly $\mathcal{R}$ can contain at most the single pair (X, Y).

If $X \in V_U$ and $Y \xrightarrow{w} \varepsilon$ and $\mathcal{R} = \left\{ (X, Y\beta_i) \,:\, i \in I \right\}$ then for each $i \in I$ we must have that $X \xrightarrow{w} \alpha_i$ such that $\alpha_i \sim \beta_i$. But then by image-finiteness there can be only a finite number of non-bisimilar such β_i.

Suppose then that $X, Y \in V_N$ and that $\mathcal{R} = \left\{ (X\alpha_i, Y\beta_i) \,:\, i \in I \right\}$ is infinite. Without loss of generality, assume that $n(Y) \leq n(X)$, and that $Y \xrightarrow{w} \varepsilon$ with $\text{length}(w) = n(Y)$. Then for each $i \in I$ we must have that $X \xrightarrow{w} \gamma_i$ such that $\gamma_i \alpha_i \sim \beta_i$. By image-finiteness, we can have only finitely many such γ_i, so we must have that $X \xrightarrow{w} \gamma$ for some γ such that $\gamma\alpha_i \sim \beta_i$ holds for infinitely many $i \in I$; by distinctness these α_is must all be non-bisimilar. For these $i \in I$ we must then have that $X\alpha_i \sim Y\gamma\alpha_i$, But then by Lemma 29 we must have that $X \sim Y\gamma$, contradicting non-decomposability. □

We are now ready to demonstrate the main result, that there is a finite relation $\mathcal{R}$ satisfying $\overset{\mathcal{R}}{\equiv} = \sim$. This will be done by induction using the following well-founded ordering $\preceq$.

Definition 33 *Define the* finite prefix norm $n_f(\alpha)$ *of* $\alpha \in V^*$ *as follows:*

$$n_f(\alpha) \;=\; \max\left\{ n(\beta) \,:\, n(\beta) < \infty \text{ and } \alpha = \beta\gamma \text{ for some } \gamma \right\}.$$

We then define the following preorder on pairs: $(\alpha_1, \alpha_2) \preceq (\beta_1, \beta_2)$ *iff* $\max\left(n_f(\alpha_1), n_f(\alpha_2) \right) \leq \max\left(n_f(\beta_1), n_f(\beta_2) \right)$.

Lemma 34 *Let* $\mathcal{R}_0 = \left\{ (X, \alpha) \,:\, X \in V_N \text{ and } X \sim \alpha \right\}$*, and let* $\mathcal{R}_1$ *be the largest set of the form*

$$\left\{ (X\alpha, Y\beta) \,:\, X\alpha, Y\beta \in V_N^* V_U \cup V_N^*,\ X\alpha \sim Y\beta, \right.$$
$$\left. \text{and } (X\alpha, Y\beta) \text{ is not decomposable} \right\}$$

which contains only distinct pairs, and containing minimal elements with respect to $\preceq$*. Then* $\mathcal{R} = \mathcal{R}_0 \cup \mathcal{R}_1$ *is a finite relation satisfying* $\overset{\mathcal{R}}{\equiv} = \sim$.

Proof Firstly, $\mathcal{R}_0$ and $\mathcal{R}_1$ must both be finite. Also we must have $\overset{\mathcal{R}}{\equiv} \subseteq \sim$. We will demonstrate by induction on $\preceq$ that $X\alpha \sim Y\beta$ implies $X\alpha \overset{\mathcal{R}}{\equiv} Y\beta$

If $(X\alpha, Y\beta)$ is decomposable, then $X, Y \in V_N$ and (without loss of generality) assume that $X \sim Y\gamma$ and $\gamma\alpha \sim \beta$. Then $n_f(\gamma\alpha) < n_f(Y\gamma\alpha) = n_f(X\alpha)$ and $n_f(\beta) < n_f(Y\beta)$, so $(\gamma\alpha, \beta) \preceq (X\alpha, Y\beta)$. Hence by induction $\gamma\alpha \overset{\mathcal{R}}{\equiv} \beta$. Then from $(X, Y\gamma) \in \mathcal{R}_0$ we get $X\alpha \overset{\mathcal{R}}{\equiv} Y\gamma\alpha \overset{\mathcal{R}}{\equiv} Y\beta$.

Suppose then that $(X\alpha, Y\beta)$ is not decomposable. Then $(X\alpha', Y\beta') \in \mathcal{R}_1$ for some $\alpha' \sim \alpha$ and $\beta' \sim \beta$ with $(\alpha', \beta') \preceq (\alpha, \beta)$.

- If $X, Y \in V_N$, then $(\alpha, \beta), (\alpha', \beta') \prec (X\alpha, Y\beta)$, so $(\alpha, \alpha'), (\beta, \beta') \prec (X\alpha, Y\beta)$. Thus by induction $\alpha \stackrel{\mathcal{R}}{\equiv} \alpha'$ and $\beta \stackrel{\mathcal{R}}{\equiv} \beta'$, so $X\alpha \stackrel{\mathcal{R}}{\equiv} X\alpha' \mathcal{R} Y\beta' \stackrel{\mathcal{R}}{\equiv} Y\beta$.
- If $X \in V_N$ and $Y \in V_U$, then $\beta = \beta' = \varepsilon$ and $X\alpha \sim Y$. Also $n_f(\alpha') \leq n_f(\alpha) < n_f(X\alpha)$, so $(\alpha, \alpha') \prec (X\alpha, Y)$. Thus by induction $\alpha \stackrel{\mathcal{R}}{\equiv} \alpha'$, so $X\alpha \stackrel{\mathcal{R}}{\equiv} X\alpha' \stackrel{\mathcal{R}}{\equiv} Y$. A symmetric argument applies for the case when $X \in V_U$ and $Y \in V_N$.
- If $X, Y \in V_U$, then $\alpha = \alpha' = \beta = \beta' = \varepsilon$ and $(X, Y) \in \mathcal{R}_1$, so $X\alpha \stackrel{\mathcal{R}}{\equiv} Y\beta$.

□

With this, we have demonstrated the desired result.

Theorem 35 *Bisimulation equivalence is decidable for BPA.*

2.1.3 A Finite Bisimulation Base for BPP

The finite bisimulation base for BPP which we describe here is taken from Hirshfeld [67], which is itself a revised presentation using Dickson's Lemma (Lemma 3) of the result of Redie [129] that every congruence on a finitely generated commutative semigroup (such as bisimilarity over BPP) is finitely generated.

Firstly, assuming that the set of variables is $V = \{X_1, \ldots, X_n\}$, we note that a state α is a monomial $X_1^{k_1} \cdots X_n^{k_n}$ over V. We then define the following two orderings on states.

Definition 36

- $X_1^{k_1} \cdots X_n^{k_n} < X_1^{l_1} \cdots X_n^{l_n}$ iff for some i: $k_i < l_i$ and $k_j = l_j$ for all $j < i$.
- $X_1^{k_1} \cdots X_n^{k_n} \sqsubset X_1^{l_1} \cdots X_n^{l_n}$ iff for some i: $k_i < l_i$ and $k_j \leq l_j$ for all j.

Note that $\alpha \sqsubset \beta$ implies that $\alpha < \beta$, and that these orderings are both well-founded: we cannot have an infinite decreasing sequence of states with respect to either ordering. Furthermore, $<$ is a total ordering: for any $\alpha \neq \beta$, either $\alpha < \beta$ or $\beta < \alpha$. Finally, both orderings are precongruences: if $\alpha < \beta$ and $\alpha' < \beta'$ ($\alpha \sqsubset \beta$ and $\alpha' \sqsubset \beta'$) then $\alpha\alpha' < \beta\beta'$ ($\alpha\alpha' \sqsubset \beta\beta'$).

Definition 37 A pair (α, β) of states is *minimal* iff $\alpha \neq \beta$, and for any $(\alpha', \beta') \sqsubset (\alpha, \beta)$ with $\alpha' \sim \beta'$ we have $\alpha' = \beta'$. (By $(\alpha', \beta') \sqsubset (\alpha, \beta)$ we mean $(\alpha', \beta') \neq (\alpha, \beta)$ but $\alpha' \sqsubseteq \alpha$ and $\beta' \sqsubseteq \beta$; note that this implies that $\alpha'\beta' \sqsubset \alpha\beta$.)

Let $\mathcal{R}$ be the collection of all minimal bisimilar pairs.

Lemma 38 *$\mathcal{R}$ is finite.*

Proof Suppose we have an infinite sequence of distinct pairs $(\alpha_i, \beta_i) \in \mathcal{R}$. We can take an infinite subsequence in which the exponents of the variable X_1 in the states α_i form a nondecreasing sequence, and then we can take a further infinite subsequence in which the exponents of the variable X_2 in the states α_i form a nondecreasing sequence, and continue in this fashion so that we are left with an infinite subsequence $(\alpha_i, \beta_i) \in \mathcal{R}$ with $(\alpha_1, \beta_1) \sqsubset (\alpha_2, \beta_2) \sqsubset \cdots$. But then (α_i, β_i) (for $i > 1$) cannot be minimal, contradicting its inclusion in $\mathcal{R}$. □

Lemma 39 $\overset{\mathcal{R}}{\equiv} = \sim$.

Proof If $\alpha \overset{\mathcal{R}}{\equiv} \beta$ then clearly $\alpha \sim \beta$ since $\sim$ is a congruence and $\mathcal{R}$ by definition only contains bisimilar pairs.

Suppose then that $\alpha \sim \beta$ but that $\alpha \not\overset{\mathcal{R}}{\equiv} \beta$ (in particular, $\alpha \neq \beta$), and that this is a minimal counterexample in that if $\alpha' \sim \beta'$ and $\alpha'\beta' < \alpha\beta$ then $\alpha' \overset{\mathcal{R}}{\equiv} \beta'$.

Since $(\alpha, \beta) \notin \mathcal{R}$ there must be $(\alpha', \beta') \sqsubset (\alpha, \beta)$ with $\alpha' \sim \beta'$ and $\alpha' \neq \beta'$. Then $\alpha'\beta' \sqsubset \alpha\beta$, so $\alpha'\beta' < \alpha\beta$, and hence by the minimality assumption we have that $\alpha' \overset{\mathcal{R}}{\equiv} \beta'$.

Without loss of generality, assume $\alpha' < \beta'$. Let β'' be such that $\beta = \beta'\beta''$. Then $\alpha \sim \beta = \beta'\beta'' \sim \alpha'\beta''$. But $\alpha\alpha'\beta'' < \alpha\beta'\beta'' = \alpha\beta$, so $\alpha \overset{\mathcal{R}}{\equiv} \alpha'\beta'' \overset{\mathcal{R}}{\equiv} \beta'\beta'' = \beta$, contradicting the assumption that $\alpha \not\overset{\mathcal{R}}{\equiv} \beta$. □

Hence we have again demonstrated the desired result.

Theorem 40 *Bisimulation equivalence is decidable for BPP.*

2.2 Polynomial Time Algorithms for BPA and BPP

In Section 2.1 we demonstrated the decidability of bisimulation equivalence over both BPA and BPP. However, the computational complexity of the algorithms which we presented shows them to be of little practical value. To overcome this deficiency we concentrate in this section on developing efficient algorithms for deciding bisimilarity within these classes of transition systems. What we demonstrate in fact are polynomial algorithms for the problem of deciding equivalences over the subclasses of normed transition systems. These algorithms will both be based on an exploitation of the decomposition properties enjoyed by normed transition systems; however, despite the apparent similarity of the two problems, different methods appear to be required.

For the algorithms, we fix a normed type 2 rewrite transition system with variable set V. The problem then is to determine efficiently—that is, in time which is polynomial in the the number n of symbols in the rewrite rules—whether or not $\alpha \sim \beta$ for $\alpha, \beta \in V^*$, where we interpret the transition system first as a BPA and then as a BPP. The first algorithm we describe for BPA is taken from Hirshfeld, Jerrum and Moller [69, 70]. The second one for BPP is taken from Hirshfeld, Jerrum and Moller [71].

2.2.1 A Polynomial Time Algorithm for Normed BPA

The basic idea behind the algorithm we present here for deciding bisimilarity between normed BPA states is to exploit the unique prime decomposition theorem by decomposing states sufficiently far to be able to establish or refute the equivalence we are considering. Further, we try to construct these decompositions by a refinement process which starts with an overly generous collection of candidate decompositions. As the algorithm progresses, invalid decompositions will gradually be weeded out.

Assume that the variables V are ordered by non-decreasing norm, so that $X < Y$ implies $n(X) \leq n(Y)$. A *base* is a set $\mathcal{B}$ of pairs $(Y, X\alpha)$, where $X, Y \in V$, $\alpha \in V^*$, $X \leq Y$ and $n(Y) = n(X\alpha)$. We insist that $\mathcal{B}$ contains at most one pair of the form $(Y, X\alpha)$ for each choice of variables X and Y, so that the cardinality of $\mathcal{B}$ is at most $O(n^2)$. A base $\mathcal{B}$ is *full* iff whenever $Y \sim X\beta$ with $Y \geq X$ there exists a pair $(Y, X\alpha) \in \mathcal{B}$ such that $\alpha \sim \beta$. In particular, $(X, X) \in \mathcal{B}$ for all $X \in V$. The key observation is that bisimilarity is the congruence closure of a finite base.

At the heart of the algorithm is the definition of a polynomial-time (in the size of $\mathcal{B}$) relation $\equiv_{\mathcal{B}} \subseteq \overset{\mathcal{B}}{\equiv}$ which satisfies $\sim \subseteq \equiv_{\mathcal{B}}$ whenever $\mathcal{B}$ is full. Central to the definition of the relation $\equiv_{\mathcal{B}}$ is the idea of a decomposing function. A function $g : V \to V^*$ is a *decomposing function* if either $g(X) = X$ or $g(X) = X_1 X_2 \cdots X_p$ with $X_i < X$ for each $1 \leq i \leq p$. Such a function g can be extended to the domain V^* in the obvious fashion by defining $g(\varepsilon) = \varepsilon$ and $g(X\alpha) = g(X)g(\alpha)$. We then define $g^*(\alpha)$ for $\alpha \in V^*$ to be the limit of $g^t(\alpha)$ as $t \to \infty$; owing to the restricted form of g we know that it must be *eventually idempotent*, that is, that this limit must exist. The notation $g[X \mapsto \alpha]$ will be used to denote the function that agrees with g at all points in V except X, where its value is α.

The definition of the relation $\equiv_{\mathcal{B}}$ may now be given. For base $\mathcal{B}$ and decomposing function g, the relation $\alpha \equiv_{\mathcal{B}}^{g} \beta$ is defined by the following decision procedure:

- if $g^*(\alpha) = g^*(\beta)$ then return true;
- otherwise let X and Y (with $X < Y$) be the leftmost mismatching pair of symbols in the words $g^*(\alpha)$ and $g^*(\beta)$;
  - if $(Y, X\gamma) \in \mathcal{B}$ then return $\alpha \equiv_{\mathcal{B}}^{g'} \beta$, where $g' = g[Y \mapsto X\gamma]$;
  - otherwise return false.

Finally, $\equiv_{\mathcal{B}}$ is defined to be $\equiv_{\mathcal{B}}^{Id}$ where Id is the identity function. (Note that this procedure terminates, as it calls itself at most $|V|$ times, each time updating the value of the parameter function g to decompose another variable.)

Lemma 41 *$\equiv_{\mathcal{B}} \subseteq \overset{\mathcal{B}}{\equiv}$ and $\sim \subseteq \equiv_{\mathcal{B}}$ whenever $\mathcal{B}$ is full.*

Proof The first inclusion is easily confirmed, since for any g constructed by the algorithm for computing $\equiv_{\mathcal{B}}$, it is the case that $X \overset{\mathcal{B}}{\equiv} g(X)$ for each $X \in V$.

For the second inclusion, suppose that $\alpha \sim \beta$ and at some point in the procedure for deciding $\alpha \equiv_{\mathcal{B}} \beta$ we have that $\mathsf{g}^*(\alpha) \neq \mathsf{g}^*(\beta)$, and that we have only ever updated g with mappings $\mathsf{X} \mapsto \gamma$ satisfying $\mathsf{X} \sim \gamma$. Let X and Y (with $\mathsf{X} < \mathsf{Y}$) be the leftmost mismatching pair. Then $\mathsf{Y} \sim \mathsf{X}\gamma$ must hold for some γ, and so, by fullness, $(\mathsf{Y}, \mathsf{X}\gamma) \in \mathcal{B}$ for some γ with $\mathsf{Y} \sim \mathsf{X}\gamma$. So the procedure does not terminate with a false result, but instead updates g with this new semantically sound mapping and continues. □

To demonstrate that $\equiv_{\mathcal{B}}$ can be decided efficiently, we are left with the problem of deciding $\mathsf{g}^*(\alpha) = \mathsf{g}^*(\beta)$ in time polynomial in the size of $\mathcal{B}$ (and finding the leftmost mismatch in case of inequality); all other elements in the definition of $\equiv_{\mathcal{B}}$ are algorithmically undemanding. Note that the words $\mathsf{g}^*(\alpha)$ and $\mathsf{g}^*(\beta)$ will in general be of exponential length, so we cannot afford to compute them explicitly. However, the relation can indeed be computed in polynomial time with a dynamic programming algorithm which exploits standard string alignments tricks based on ideas from Knuth, Morris and Pratt [94]. The details of this algorithm can be found in [70].

The main task now is to discover a small full base that contains only semantically sound decomposition pairs. To do this, we start with a small full base, and then proceed to refine the base iteratively whilst maintaining fullness. Informally, we are proposing that at any instant the current base should consist of pairs (X, α) representing candidate decompositions, that is, pairs such that the relationship $\mathsf{X} \sim \alpha$ is consistent with information gained so far. The refinement step is as follows.

Given a base $\mathcal{B}$, define the sub-base $\widehat{\mathcal{B}} \subseteq \mathcal{B}$ to be the set of pairs $(\mathsf{X}, \alpha) \in \mathcal{B}$ such that

- if $\mathsf{X} \xrightarrow{\mathsf{a}} \beta$ then $\alpha \xrightarrow{\mathsf{a}} \gamma$ with $\beta \equiv_{\mathcal{B}} \gamma$, and
- if $\alpha \xrightarrow{\mathsf{a}} \gamma$ then $\mathsf{X} \xrightarrow{\mathsf{a}} \beta$ with $\beta \equiv_{\mathcal{B}} \gamma$.

Lemma 42 *If $\mathcal{B}$ is full then $\widehat{\mathcal{B}}$ is full.*

Proof Suppose $\mathsf{Y} \sim \mathsf{X}\beta$ with $\mathsf{Y} \geq \mathsf{X}$. By fullness of $\mathcal{B}$, there exists a pair $(\mathsf{Y}, \mathsf{X}\alpha) \in \mathcal{B}$ such that $\alpha \sim \beta$. We show that the pair $(\mathsf{Y}, \mathsf{X}\alpha)$ survives the refinement step, to be included in $\widehat{\mathcal{B}}$. Note that, since $\sim$ is a congruence, $\mathsf{Y} \sim \mathsf{X}\alpha$. Thus, if $\mathsf{Y} \xrightarrow{\mathsf{a}} \gamma$ then $\mathsf{X}\alpha \xrightarrow{\mathsf{a}} \delta$ for some δ satisfying $\delta \sim \gamma$. By fullness of $\mathcal{B}$ and Lemma 41, $\delta \equiv_{\mathcal{B}} \gamma$. Similarly, if $\mathsf{X}\alpha \xrightarrow{\mathsf{a}} \delta$ then $\mathsf{Y} \xrightarrow{\mathsf{a}} \gamma$ with $\gamma \sim \delta$, and hence $\gamma \equiv_{\mathcal{B}} \delta$. The pair $(\mathsf{Y}, \mathsf{X}\alpha)$ therefore satisfies the conditions for inclusion in $\widehat{\mathcal{B}}$. □

In general, the refinement step makes progress, i.e., the new base $\widehat{\mathcal{B}}$ is strictly contained in the base $\mathcal{B}$ from which it was derived. If, however, no progress occurs, an important deduction may be made.

Lemma 43 *If $\widehat{\mathcal{B}} = \mathcal{B}$ then $\equiv_{\mathcal{B}} \subseteq \sim$.*

Proof The relation $\equiv_{\mathcal{B}}$ is contained in $\overset{\mathcal{B}}{\equiv}$, the congruence closure of $\mathcal{B}$; and if $\widehat{\mathcal{B}} = \mathcal{B}$ then $\mathcal{B}$ must be a bisimulation base, in which case by Theorem 25, $\equiv_{\mathcal{B}} \subseteq \overset{\mathcal{B}}{\equiv} \subseteq \sim$. □

Note that by iteratively applying the refinement step $\mathcal{B} := \widehat{\mathcal{B}}$ to a full initial base, we are guaranteed by the preceding three lemmas to stabilise at some full base $\mathcal{B}$ for which $\equiv_{\mathcal{B}} = \sim$.

Finally, we are left with the task of constructing the initial base $\mathcal{B}_0$. This is achieved as follows. For each $X \in V$ and each $0 \leq \nu \leq n(X)$, let $[X]_\nu$ be some state that can be reached from X via a sequence of ν norm-reducing transitions. (Note that some norm-reducing transition is available to every state.)

Lemma 44 *The base* $\mathcal{B}_0 = \left\{ \left(Y, X[Y]_{n(X)}\right) : X, Y \in V \text{ and } X \leq Y \right\}$ *is full.*

Proof Suppose $Y \sim X\beta$ with $X \leq Y$, and let $\nu = n(X)$; then $(Y, X[Y]_\nu) \in \mathcal{B}_0$ for some $[Y]_\nu$ such that $Y \xrightarrow{s} [Y]_\nu$ in ν norm-reducing steps, where $s \in A^\nu$. But the norm-reducing sequence $Y \xrightarrow{s} [Y]_\nu$ can only be matched by $X\beta \xrightarrow{s} \beta$. Hence $[Y]_\nu \sim \beta$, and $\mathcal{B}_0$ must be full. □

The basic structure of the procedure for deciding bisimilarity between normed states α and β is now clear: simply iterate the refinement procedure $\mathcal{B} := \widehat{\mathcal{B}}$ from the initial base $\mathcal{B} = \mathcal{B}_0$ until it stabilises at the desired base $\mathcal{B}$, and then test $\alpha \equiv_{\mathcal{B}} \beta$. By the preceding four lemmas, this test is equivalent to $\alpha \sim \beta$.

We have not been specific about which state $[X]_\nu$ is to be selected among those reachable from X via a sequence of ν norm-reducing transitions. A suitable choice is provided by the following recursive definition. For each variable $X \in V$, let $\alpha_X \in V^*$ be some state reachable from X by a single norm-reducing transition $X \xrightarrow{a} \alpha_X$. Then,

$$\begin{aligned} [\alpha]_0 &= \alpha, \\ [X\beta]_p &= \begin{cases} [\beta]_{p-n(X)}, & \text{if } p \geq n(X); \\ [\alpha_X]_{p-1}\beta, & \text{if } p < n(X). \end{cases} \end{aligned}$$

Lemma 45 *With this definition for* $[\cdot]_\nu$*, the base* $\mathcal{B}_0$ *introduced in Lemma 44 may be explicitly constructed in polynomial time; in particular, every pair in* $\mathcal{B}_0$ *has a compact representation as an element of* $V \times V^*$.

Proof It is easily checked that the natural recursive algorithm based on the definition is polynomial-time bounded. □

We have already observed that $\mathcal{B}_0$ contains $O(n^2)$ pairs, so the refinement procedure is iterated at most $O(n^2)$ times. Hence we have succeeded in demonstrating our result.

Theorem 46 *There is a polynomial-time (in the lengths of the states* α *and* β*, and the size of the rewrite rules) procedure for deciding bisimilarity between two states* α *and* β *of a normed type 2 sequential rewrite transition system.*

Simple context-free grammars

Recall that a simple grammar is a context-free grammar in Greibach normal form such that for every variable X and terminal symbol a there is at most one production of the form $X \to a\alpha$; these correspond to deterministic type 2 sequential rewrite transition systems. The decision procedure given by Korenjak and Hopcroft [95] for deciding language equivalence between simple grammars is doubly exponential; this time complexity was improved by Caucal [30] to be singly exponential. The following result demonstrates that this problem has a polynomial-time solution.

Theorem 47 *There is a polynomial-time algorithm for deciding equivalence of simple grammars.*

Proof To obtain a polynomial-time decision procedure for deciding language equivalence of simple context-free grammars (deterministic type 2 sequential rewrite transition systems), we merely recall from Lemma 17 that in the case of normed deterministic transition systems, language equivalence and bisimulation equivalence coincide. We can restrict attention to normed grammars, as any unnormed grammar can be transformed into a language-equivalent normed grammar by removing productions containing unnormed nonterminals. (Note that this transformation does not preserve bisimulation equivalence, which makes it inapplicable for reducing the unnormed case to the normed case in checking bisimilarity.) Thus language equivalence of simple grammars may be checked in polynomial time by the procedure presented in the previous two sections. □

2.2.2 A Polynomial Time Algorithm for Normed BPP

To demonstrate that we can decide bisimilarity between normed BPP in polynomial time, we require a vastly different technique than that used for the BPA case; nonetheless the technique still relies completely on the unique factorisation property.

To start off, we assume without loss of generality that the variables are given in order of non-decreasing norm, so that $n(X_1) \leq n(X_2) \leq \cdots \leq n(X_n)$. Define the *size* of a monomial $\alpha \in V^*$ to be the sum of the lengths of the binary encodings of the various exponents appearing in the monomial; the size of a production $X \xrightarrow{a} \beta$ to be the length of the triple (X, a, β), encoded in binary; and the size of a rewrite transition system to be the sum of the sizes of all the rewrite rules contained within it.

To prepare for the description of the algorithm, and the proof that it runs in polynomial time, we require some definitions and a few preparatory lemmas. We omit the (generally straightforward) proofs of these lemmas; they can be found in [71].

Suppose $\mathcal{R}$ is any relation on V^*. We say that a pair $(\alpha, \beta) \in V^* \times V^*$ *satisfies (norm-reducing) expansion in* $\mathcal{R}$ if

- if $\alpha \xrightarrow{a} \alpha'$ is a (norm-reducing) transition then $\beta \xrightarrow{a} \beta'$ for some β' with $\alpha' \mathcal{R} \beta'$;

and

- if $\beta \xrightarrow{a} \beta'$ is a (norm-reducing) transition then $\alpha \xrightarrow{a} \alpha'$ for some α' with $\alpha' \mathcal{R} \beta'$.

Observe that a relation $\mathcal{R}$ is a bisimulation if every pair $(\alpha, \beta) \in \mathcal{R}$ satisfies expansion in $\sim$. Observe also that if $\mathcal{R}$ is an equivalence relation (respectively, congruence) then the relations "satisfies expansion in $\mathcal{R}$" and "satisfies norm-reducing expansion in $\mathcal{R}$" are both equivalence relations (respectively, congruences).

Define a *unique decomposition base*, $\mathcal{D}$, to be a pair (Π, Γ), where $\Pi = \Pi(\mathcal{D}) = \{\mathsf{P}_1, \ldots, \mathsf{P}_r\} \subseteq \mathsf{V}$ is a set of *primes*, and $\Gamma = \Gamma(\mathcal{D})$ is a set of pairs $(\mathsf{X}, \mathsf{P}_1^{x_1} \ldots \mathsf{P}_r^{x_r})$, one for each non-prime variable $\mathsf{X} \in \mathsf{V} - \Pi$. The set Γ may be viewed as specifying, for each non-prime variable X, a decomposition of X into primes. (Note that these "primes" are not in general the primes with respect to the maximal bisimulation, which were the subject of Theorem 22.) A unique decomposition base defines an equivalence relation $\equiv_{\mathcal{D}}$ on V^*: the relation $\alpha \equiv_{\mathcal{D}} \beta$ holds between $\alpha, \beta \in \mathsf{V}^*$ if the prime decompositions of α and β are equal (as monomials).

Lemma 48 *Let $\mathcal{D}$ be a unique decomposition base. Then:*

(i) *the equivalence relation $\equiv_{\mathcal{D}}$ is a congruence with cancellation (that is, $\alpha \equiv_{\mathcal{D}} \beta$ whenever $\alpha\gamma \equiv_{\mathcal{D}} \beta\gamma$) which coincides with $\overset{\mathcal{D}}{\equiv}$, the smallest congruence containing $\Gamma(\mathcal{D})$;*

(ii) *there is a polynomial-time (in the size of α and β) algorithm to decide $\alpha \equiv_{\mathcal{D}} \beta$ for arbitrary $\alpha, \beta \in \mathsf{V}^*$;*

(iii) *the relation $\equiv_{\mathcal{D}}$ is a bisimulation provided every pair in $\Gamma(\mathcal{D})$ satisfies expansion within $\equiv_{\mathcal{D}}$; this condition may be checked by a polynomial-time algorithm;*

(iv) *the maximal bisimulation $\sim$ coincides with the congruence $\equiv_{\mathcal{D}}$, where $\mathcal{D}$ represents the unique decomposition in $\sim$.*

The next lemma allows us to shrink a congruence, defined by a unique decomposition base, whenever it is strictly larger than the maximal bisimulation.

Lemma 49 *Let $\mathcal{D}$ be a unique decomposition base such that the congruence $\equiv_{\mathcal{D}}$ is norm-preserving and strictly contains the maximal bisimulation $\sim$. Define the relation $\equiv$ as follows: for all $\alpha, \beta \in \mathsf{V}^*$, the relationship $\alpha \equiv \beta$ holds iff $\alpha \equiv_{\mathcal{D}} \beta$ and the pair (α, β) satisfies expansion in $\equiv_{\mathcal{D}}$. Then it is possible, in polynomial time, to find (a representation of) a relation $\equiv$ on V^* such that:*

(i) *the relation $\alpha \equiv \beta$ is decidable in polynomial time (in the sum of the sizes of α and β);*

(ii) *the relation $\equiv$ is a congruence;*

(iii) *there is a variable* $X \in V$ *that is decomposable in* $\equiv_{\mathcal{D}}$ *but not in* $\equiv$*;*

(iv) *the inclusions* $\sim \subseteq \equiv \subset \equiv_{\mathcal{D}}$ *hold.*

The final lemma allows us to "smooth out" an unmanageable congruence into a congruence defined by a unique decomposition base.

Lemma 50 *Let* $\equiv$ *be a norm-preserving, polynomial-time computable congruence satisfying* $\sim \subseteq \equiv$*, where* $\sim$ *denotes maximal bisimulation. Then there is a decomposition base* $\mathcal{D}$*, computable in polynomial time, such that* $\sim \subseteq \equiv_{\mathcal{D}} \subseteq \equiv$*.*

With the three preceding lemmas in place, the procedure for deciding bisimulation equivalence is clear.

(1) Let the congruence $\equiv$ be defined by $\alpha \equiv \beta$ iff $n(\alpha) = n(\beta)$.

(2) Compute a decomposition base $\mathcal{D}$ with $\sim \subseteq \equiv_{\mathcal{D}} \subseteq \equiv$, using Lemma 50.

(3) If $\equiv_{\mathcal{D}}$ is a bisimulation—a condition that can be checked in polynomial time using Lemma 48—then halt and return the relation $\equiv_{\mathcal{D}}$.

(4) Compute a congruence $\equiv$ satisfying $\sim \subseteq \equiv \subset \equiv_{\mathcal{D}}$, using Lemma 49. Go to step 2.

The main result is then virtually immediate.

Theorem 51 *Given a normed BPP over variable set* V*, there is a polynomial-time (in the size of* α*,* β*, and the transition rules) algorithm to decide* $\alpha \sim \beta$ *for arbitrary* $\alpha, \beta \in V^*$*.*

Proof On each iteration of the loop formed by lines (2)–(4), the number of primes increases by at least one. Thus the number of iterations is bounded by n, and each iteration requires only polynomial time by the three preceding lemmas. □

2.3 Computing a Finite Bisimulation Base for BPA

In Section 2.1.2, we demonstrated the existence of a finite bisimulation base for BPA; this permitted us to infer the decidability of bisimilarity between BPA states using two semi-decision procedures. In this section, we describe an algorithm for constructing a general bisimulation base, a finite relation $\mathcal{R}$ in which the least congruence $\overset{\mathcal{R}}{\equiv}$ containing $\mathcal{R}$ is equal to $\sim$. From this we deduce a stronger result than the decidability of bisimulation: the largest bisimulation $\sim$ is an effective rational relation. Furthermore, the algorithm that we describe will demonstrate that the equivalence problem with respect to bisimilarity of BPA states can be solved in double exponential time.

Given any binary relation $\mathcal{R}$ on V^*, we define $E(\mathcal{R})$ to be the set of pairs in $\mathcal{R}$ which satisfy expansion in $\overset{\mathcal{R}}{\equiv}$; that is, $(\alpha, \beta) \in E(\mathcal{R})$ iff $(\alpha, \beta) \in \mathcal{R}$ and

- if $\alpha \stackrel{a}{\longrightarrow} \alpha'$ then $\beta \stackrel{a}{\longrightarrow} \beta'$ for some β' with $\alpha' \stackrel{\mathcal{R}}{\equiv} \beta'$; and
- if $\beta \stackrel{a}{\longrightarrow} \beta'$ then $\alpha \stackrel{a}{\longrightarrow} \alpha'$ for some α' with $\alpha' \stackrel{\mathcal{R}}{\equiv} \beta'$.

In particular, $\mathcal{R}$ is a bisimulation base (cf. Definition 24) if $\mathcal{R} = \mathsf{E}(\mathcal{R})$, in which case $\stackrel{\mathcal{R}}{\equiv}$ is a bisimulation (cf. Lemma 25). Our intention is to construct a finite base $\mathcal{R}$ generating the largest bisimulation: $\sim \; = \stackrel{\mathcal{R}}{\equiv}$. To do this, we mimic the construction from Section 2.2.1:

1. we find a *finite* relation $\mathcal{R}_0$ which is *complete*, meaning that $\sim \; = \stackrel{\mathcal{R}_0 \cap \sim}{\equiv}$ (this is a refinement of the notion of *fullness* defined in Section 2.2.1);
2. we then repeatedly apply E to compute $\mathcal{R}_0 \supseteq \mathsf{E}(\mathcal{R}_0) \supseteq \mathsf{E}^2(\mathcal{R}_0) \supseteq \cdots$ until we find that $\mathsf{E}^n(\mathcal{R}_0) = \mathsf{E}^{n+1}(\mathcal{R}_0)$; we then take $\mathcal{R} = \mathsf{E}^n(\mathcal{R}_0)$. (Note that n can be no greater than the size of $\mathcal{R}_0$.)

That this procedure provides the desired finite base $\mathcal{R}$ satisfying $\sim \; = \stackrel{\mathcal{R}}{\equiv}$ follows from the fact that completeness is preserved by the application of E, as demonstrated in the next result (cf. Lemma 42).

Lemma 52 *If $\mathcal{B}$ is complete, then $\mathcal{B} \cap \sim \; = \; \mathsf{E}(\mathcal{B}) \cap \sim$, so $\mathsf{E}(\mathcal{B})$ is complete.*

Proof Assume that $\mathcal{B}$ is complete, so that $\sim \; = \stackrel{\mathcal{B} \cap \sim}{\equiv} \; \subseteq \stackrel{\mathcal{B}}{\equiv}$. Then since E is monotonic, we get $\sim \; = \mathsf{E}(\sim) \; \subseteq \; \mathsf{E}(\stackrel{\mathcal{B}}{\equiv})$, and hence

$$\begin{aligned}
\mathcal{B} \cap \sim \; &= \; \mathcal{B} \cap \mathsf{E}(\stackrel{\mathcal{B}}{\equiv}) \cap \sim && \Big(\text{since} \sim \; \subseteq \; \mathsf{E}(\stackrel{\mathcal{B}}{\equiv})\Big) \\
&= \; \mathcal{B} \cap \mathsf{E}(\mathcal{B}) \cap \sim && \Big(\text{since } \mathcal{B} \cap \mathsf{E}(\stackrel{\mathcal{B}}{\equiv}) \; = \; \mathcal{B} \cap \mathsf{E}(\mathcal{B})\Big) \\
&= \; \mathsf{E}(\mathcal{B}) \cap \sim && \Big(\text{since } \mathsf{E}(\mathcal{B}) \; \subseteq \; \mathcal{B}\Big).
\end{aligned}$$

□

We shall apply this method to an arbitrary BPA system. As before we split the set of variables into two disjoint sets $\mathsf{V} = \mathsf{V}_\mathsf{N} \cup \mathsf{V}_\mathsf{u}$ with the variables in V_N having finite norm and those in V_u having infinite norm, and make use of the *finite prefix norm* from Definition 33:

$$\mathrm{n_f}(\alpha) \;\; = \;\; \max\Big\{ \; \mathrm{n}(\beta) \; : \; \mathrm{n}(\beta) < \infty \text{ and } \alpha = \beta\gamma \text{ for some } \gamma \; \Big\}.$$

Given the following three relations

$$\begin{aligned}
\mathcal{R}_0 \;\; &= \;\; \Big\{ \; (A, AB) \; : \; A \in \mathsf{V}_\mathsf{u} \text{ and } B \in \mathsf{V} \; \Big\} \\
\mathcal{R}_\mathsf{f} \;\; &= \;\; \Big\{ \; (A, \alpha) \in \mathsf{V}_\mathsf{N} \times \mathsf{V}_\mathsf{N}^* \; : \; \mathrm{n}(A) = \mathrm{n}(\alpha) \; \Big\} \\
\mathcal{R}_\infty \;\; &= \;\; \mathsf{V}_\mathsf{N}^* \mathsf{V}_\mathsf{u} \;\; \times \;\; \mathsf{V}_\mathsf{N}^* \mathsf{V}_\mathsf{u}
\end{aligned}$$

we shall construct a finite complete relation $\mathcal{R} = \mathcal{R}_0 \cup \mathcal{R}_\mathsf{f} \cup \mathcal{R}'$ with $\mathcal{R}' \subseteq \mathcal{R}_\infty$. Then by Lemmas 52 and 53 below, we shall be able to construct $\mathcal{R} \cap \sim$.

Lemma 53 *For any finite $\mathcal{S} \subseteq \mathcal{R}_f \cup \mathcal{R}_\infty$ with $\mathcal{R}_0 \subseteq \mathcal{S}$, the relation $\overset{\mathcal{S}}{\equiv}$ is decidable; hence we can compute $\mathsf{E}(\mathcal{S})$.*

Proof Let

$$\begin{aligned} \mathcal{S}_f &= \mathcal{S} \cap \mathcal{R}_f, \quad \text{and} \\ \mathcal{S}_\infty &= \Big\{ (\alpha', \beta') \,:\, \alpha \overset{\mathcal{S}_f}{\equiv} \alpha' \text{ and } \beta \overset{\mathcal{S}_f}{\equiv} \beta' \text{ for some } (\alpha, \beta) \in \mathcal{S} \cap \mathcal{R}_\infty \Big\}. \end{aligned}$$

As $\mathcal{S}_f$ is finite and norm-preserving, $\mathcal{S}_\infty$ must be finite as well. Now,

$$\begin{aligned} \overset{\mathcal{S}}{\equiv} &= \Big\{ (\alpha, \beta) \,:\, \alpha, \beta \in V_N^* \text{ and } \alpha \overset{\mathcal{S}_f}{\equiv} \beta \Big\} \\ &\cup \Big\{ (\alpha\gamma, \beta\delta) \,:\, \alpha, \beta \in V_N^* V_u \text{ and } \alpha \overset{\mathcal{S}_\infty}{\equiv} \beta \Big\}. \end{aligned}$$

Note that $\overset{\mathcal{S}_f}{\equiv}$ restricted to V_N^* is decidable: $\mathcal{S}_f$ is norm-preserving, and hence any equivalence class is an effective finite set. Furthermore $\overset{\mathcal{S}_\infty}{\equiv}$ restricted to $V_N^* V_u$ is decidable: it is the suffix derivation of $\mathcal{S}_\infty \cup \mathcal{S}_\infty^{-1}$, and hence any equivalence class is an effective rational set [19]. □

For any $n \geq 1$, we define the relation $\mathcal{R}_n$ as follows:

$$\mathcal{R}_n = \mathcal{R}_0 \cup \mathcal{R}_f \cup \Big\{ (\alpha, \beta) \in \mathcal{R}_\infty \,:\, n_f(\alpha), n_f(\beta) < n \Big\}.$$

We shall compute a constant e such that $\mathcal{R}_e$ is complete, after which we can compute $\mathcal{R}_e \cap \sim$ and hence infer that $\sim$ is decidable. In fact, we get a stronger result: we can construct a transducer (a finite automaton labelled by pairs of states) recognizing the bisimulation.

Theorem 54 *Bisimilarity is an effective rational relation over any BPA system.*

Proof We assume that we have our promised integer e such that $\mathcal{R}_e$ is complete. By Lemmas 52 and 53, we compute $\mathcal{R}_e \cap \sim \;=\; \mathcal{R}_0 \cup (\mathcal{R}_f \cap \sim) \cup \mathcal{S}_e$, where

$$\mathcal{S}_e = \Big\{ (\alpha, \beta) \,:\, \alpha, \beta \in V_N^* V_u, \;\; \alpha \sim \beta \;\text{ and }\; n_f(\alpha), n_f(\beta) < e \Big\}.$$

First we show that the set of bisimilar pairs of infinite norm is an effective rational relation. We have

$$\begin{aligned} \sim \cap\, V^* V_u V^* \times V^* V_u V^* &= \overset{\mathcal{R}_e \cap \sim}{\equiv} \cap\, V^* V_u V^* \times V^* V_u V^* \\ &= \Big(\overset{\mathcal{S}_e}{\equiv} \cap\, V_N^* V_u \times V_N^* V_u \Big) \cdot \Big(V^* \times V^* \Big) \end{aligned}$$

But $(\overset{\mathcal{S}_e}{\equiv} \cap\, V_N^* V_u \times V_N^* V_u)$ is the suffix derivation according to $\mathcal{S}_e$ restricted to the recognizable relation $V_N^* V_u \times V_N^* V_u$, and hence is an effective rational relation [32].

It then remains to show that the set of bisimilar pairs of finite norm is an effective rational relation. For this, we can exploit the Unique Factorisation Theorem 21. Let $V_p \subseteq V_N$ be the set of all primes without repetition up to $\sim$; that is, for every $\alpha \in V_N^*$ there is a unique $\beta \in V_p^*$ such that $\alpha \sim \beta$. Now take

$$\mathcal{S} \;=\; \Big\{\, (X,\alpha) \;:\; X \in V_N,\ \alpha \in V_p^*,\ \text{and } X \sim \alpha \,\Big\}.$$

The set of bisimilar pairs of finite norm can then be expressed as the effective rational set $\mathcal{S}^* \circ (\mathcal{S}^{-1})^*$. □

It now remains to explain the computation of the constant e such that $\mathcal{R}_e$ is complete. A useful tool is the following *divergence* function:

$$\begin{aligned} \mathrm{Div}(\alpha,\beta) \;&=\; \min\Big(\{\, n \;:\; \alpha \not\sim_n \beta \,\} \cup \{\infty\}\Big) \\ &=\; 1 + \max\{\, n \;:\; \alpha \sim_n \beta \,\}. \end{aligned}$$

An important property is that $1/\mathrm{Div}$ is an ultrametric distance.

Lemma 55

a) $1 \;\le\; \mathrm{Div}(\alpha,\beta) \;=\; \mathrm{Div}(\beta,\alpha)$;

b) $\min\Big(\mathrm{Div}(\alpha,\beta), \mathrm{Div}(\beta,\gamma)\Big) \;\le\; \mathrm{Div}(\alpha,\gamma)$;

c) *If* $\mathrm{Div}(\alpha,\beta) > \mathrm{Div}(\alpha,\gamma)$ *then* $\mathrm{Div}(\beta,\gamma) = \mathrm{Div}(\alpha,\gamma)$;

d) *If* $n(\alpha) < n(\beta)$ *then* $\mathrm{Div}(\alpha,\beta) \le 1 + n(\alpha)$;

e) *If* $\alpha \xrightarrow{u} \alpha'$ *and* $|u| < \mathrm{Div}(\alpha,\beta)$ *then* $\beta \xrightarrow{u} \beta'$ *such that* $\mathrm{Div}(\alpha,\beta) \le |u| + \mathrm{Div}(\alpha',\beta')$.

We extend Div to any binary relation R over states as follows.

$$\mathrm{minDiv}(R) \;=\; \min\Big\{\; \mathrm{Div}(\alpha,\beta) \;:\; \alpha\, R\, \beta \;\Big\}$$

With this, we may now generalize Lemma 55(c).

Lemma 56 *If* $\alpha\, R^*\, \alpha'$, $\beta\, R^*\, \beta'$, *and* $\mathrm{Div}(\alpha,\beta) < \mathrm{minDiv}(R)$ *then* $\mathrm{Div}(\alpha,\beta) = \mathrm{Div}(\alpha',\beta')$.

The above properties of the divergence function are true for arbitrary transition systems. Henceforth we consider only BPA systems, in which case the divergence function satisfies the following additional properties.

Lemma 57 *For every* $\alpha,\beta,\gamma,\delta \in V^*$, *we have the following.*

a) *If* $\gamma \sim \delta$ *then* $\mathrm{Div}(\alpha,\beta) \le \mathrm{Div}(\alpha\gamma,\beta\delta) \le \mathrm{Div}(\alpha,\beta) + n(\gamma) \le \mathrm{Div}(\gamma\alpha,\delta\beta)$.

b) *If* $\alpha\gamma \sim \beta\delta$ *and* $1 < \mathrm{Div}(\alpha,\beta) < \infty$ *then* $\alpha \xrightarrow{a} \alpha'$ *and* $\beta \xrightarrow{a} \beta'$ *such that* $\alpha'\gamma \sim \beta'\delta$ *and* $\mathrm{Div}(\alpha',\beta') < \mathrm{Div}(\alpha,\beta)$.

c) *If* $\alpha \overset{\mathcal{R}}{\equiv} \gamma$, $\beta \overset{\mathcal{R}}{\equiv} \delta$, *and* $\mathrm{Div}(\alpha,\beta) < \mathrm{minDiv}(R)$ *then* $\mathrm{Div}(\alpha,\beta) = \mathrm{Div}(\gamma,\delta)$.

If we can compute an integer $a_{\alpha,\beta}$ for any α, β such that $a_{\alpha,\beta} > Div(\alpha, \beta)$ when $\alpha \not\sim \beta$, then we would have the basis of an algorithm for deciding $\sim$. In fact, we are not able to find such a bound for an arbitrary BPA system; but for any BPA system, we shall show that there exists an equivalent BPA system in a form allowing to compute such a bound.

Consider any BPA system with rewrite rules P. For each $\alpha \in Im(P)$ with $n(\alpha) = \infty$ (that is, for each unnormed sequence α which appears as the right hand side of some rule of P) we associate a new variable X_α in such a way that $X_\alpha = X_\beta$ whenever $\alpha \sim \beta$. (This association is not effective, but we are interested only in the divergence function.) For each such variable X_α we also introduce a new label a_α. We then define a new BPA system with the following set $\widehat{P}$ of rewrite rules:

$$\begin{aligned} \widehat{P} \;&=\; \left\{ A \xrightarrow{a} \alpha \;:\; A \xrightarrow{a} \alpha \in P \text{ and } n(\alpha) < \infty \right\} \\ &\cup\; \left\{ A \xrightarrow{a} X_\alpha \,,\, X_\alpha \xrightarrow{a_\alpha} X_\alpha \;:\; A \xrightarrow{a} \alpha \in P \text{ and } n(\alpha) = \infty \right\}. \end{aligned}$$

The following then attests that these two BPA systems are equivalent.

Lemma 58 *For* $\alpha, \beta \in V_N^*$, $\alpha \sim_P \beta$ *iff* $\alpha \sim_{\widehat{P}} \beta$.

We have that $Div_{\widehat{P}}(\alpha, \beta) \leq Div_P(\alpha, \beta)$, but equality is not true in general. The new system $\widehat{P}$ has the following restriction: each right hand side is either a normed sequence or an unnormed variable. Furthermore, $Div(A, B) = 1$ for every $A, B \in V_u$ with $A \not\sim_{\widehat{P}} B$. These restrictions allow us to find an upper bound to the divergence for non-bisimilar pairs. Before proceedings, we need first to introduce some further notations. For any finite set $E \subseteq V^*$, we let $|E|$ denote its cardinality; $max\text{-}n(E) = max\{ n(\alpha) \;:\; \alpha \in E \}$; $min\text{-}n(E) = min\{ n(\alpha) \;:\; \alpha \in E \}$; $max\text{-}n_f(E) = max\{ n_f(\alpha) \;:\; \alpha \in E \}$; and $min\text{-}n_f(E) = min\{ n_f(\alpha) \;:\; \alpha \in E \}$. We shall also freely apply these max and min functions to lists as well as sets. We let $P_f = \{ A \xrightarrow{a} \alpha \in P \;:\; n(\alpha) < \infty \}$ be the restriction of a system P to its rules of finite norm. Then for any $\alpha, \beta \in V^*$, we define

$$a_{\alpha,\beta} \;=\; min\text{-}n(\alpha, \beta) \;+\; (|V_N| - 1) \cdot max\text{-}n(Im(P_f)) + max\text{-}n(V_N)$$

Lemma 59 *For any BPA system* P *and any* $\alpha, \beta \in V_N^*$, *if* $\alpha \not\sim \beta$ *then* $Div_{\widehat{P}}(\alpha, \beta) \leq a_{\alpha,\beta}$.

Proof Using Lemmas 55, 56 and 57, we generalize directly the construction given in [30] for simple grammars to obtain the same bound $a_{\alpha,\beta}$ for any BPA system in which each right hand side is either a normed sequence or an unnormed variable; and two unnormed non-bisimilar variables are of divergence one (this last condition is not strictly necessary, but without it the bound must be changed). □

For a normed system P, $\widehat{P} = P$, and $a_{\alpha,\beta}$ is optimal when $P = V \times \Sigma \times \{\varepsilon\}$, in which case $max\text{-}n(Im(P)) = 0$ and $max\text{-}n(V) = 1$, so $a_{\alpha,\beta} = min(|\alpha|, |\beta|) + 1 = Div(\alpha, \beta)$ whenever $\alpha \not\sim \beta$. This bound $a_{\alpha,\beta}$ on $Div_{\widehat{P}}(\alpha, \beta)$ also provides a bound on the length of a derivation of P from (α, β) to a pair with distinct norms, and which preserves a given unification, as formulated in the following.

Lemma 60 *For $\alpha, \beta \in V_N^*$, if $\alpha\gamma \sim \beta\delta$ and $\alpha \not\sim \beta$ then for some α', β' and u with $|u| < a_{\alpha,\beta}$ we have that $\alpha \xrightarrow{u} \alpha'$, $\beta \xrightarrow{u} \beta'$, $n(\alpha') \neq n(\beta')$, and $\alpha'\gamma \sim \beta'\delta$.*

Proof By Lemma 58, $\alpha \not\sim_{\hat{P}} \beta$, and by Lemma 59, $Div_{\hat{P}}(\alpha, \beta) \leq a_{\alpha,\beta}$. By Lemma 57, using induction on $Div_{\hat{P}}(\alpha, \beta)$, there exist two derivations

$$\alpha = \alpha_1 \xrightarrow{a_1}_{\hat{P}} \alpha_2 \xrightarrow{a_2}_{\hat{P}} \cdots \xrightarrow{a_{n-1}}_{\hat{P}} \alpha_n \qquad \text{and} \qquad \beta = \beta_1 \xrightarrow{a_1}_{\hat{P}} \beta_2 \xrightarrow{a_2}_{\hat{P}} \cdots \xrightarrow{a_{n-1}}_{\hat{P}} \beta_n$$

such that $\alpha_i\gamma \sim \beta_i\delta$ for every $1 \leq i \leq n$, and

$$Div_{\hat{P}}(\alpha_1, \beta_1) > Div_{\hat{P}}(\alpha_2, \beta_2) > \cdots > Div_{\hat{P}}(\alpha_n, \beta_n) = 1.$$

In particular, $n \leq Div_{\hat{P}}(\alpha, \beta) \leq a_{\alpha,\beta}$. Since $Div_{\hat{P}}(\alpha_n, \beta_n) = 1$ and $\alpha_n\gamma \sim \beta_n\delta$, either $\alpha_n = \varepsilon \neq \beta_n$ or $\alpha_n \neq \varepsilon = \beta_n$, so $n(\alpha_n) \neq n(\beta_n)$. Thus $m = \min\{i : n(\alpha_i) \neq n(\beta_i)\}$ exists, and we take $u = a_1 \ldots a_{m-1}$, so $|u| = m - 1 < n \leq a_{\alpha,\beta}$. By symmetry, we may assume that $n(\alpha_m) < n(\beta_m)$.

- If $n(\beta_m) < \infty$, then it suffices to take $\alpha' = \alpha_m$ and $\beta' = \beta_m$.
- If $n(\beta_m) = \infty$, then $m > 1$ and $\beta_{m-1} \in V_N^+$. Let $B\mu = \beta_{m-1}$; then $B \xrightarrow{a_{m-1}}_P \rho$ with $X_\rho\mu = \beta_m$. Thus it suffices to take $\alpha' = \alpha_m$ and $\beta' = \rho\mu$; in particular,

$$\alpha'\gamma = \alpha_m\gamma \sim \beta_m\delta = X_\rho\mu\delta \sim \rho \sim \rho\mu\delta = \beta'\delta.$$

□

If $\gamma = \delta$ in the above Lemma, then δ is *repetitive*: $\delta \sim \rho\delta$ for some $\rho \neq \varepsilon$, and we can find such a ρ with a finite norm bounded by $b_{\text{max-n}(\alpha,\beta)}$, where

$$b_n = (n + |V_N|) \cdot \text{max-n}(Im(P_f)) \cdot (\text{max-n}_f(Im(P)))^2$$

Lemma 61 *For $\alpha, \beta \in V_N^*$, if $\alpha\delta \sim \beta\delta$ and $\alpha \not\sim \beta$ then $\delta \sim \gamma\delta$ for some $\gamma \neq \varepsilon$ with $n_f(\gamma) < b_{max\text{-}n(\alpha,\beta)}$.*

Proof By Lemma 60, there are derivations $\alpha \xrightarrow{u} \alpha'$ and $\beta \xrightarrow{u} \beta'$ such that $|u| < a_{\alpha,\beta}$, $n(\alpha') \neq n(\beta')$ and $\alpha'\delta \sim \beta'\delta$. Thus there are derivations

$$\left(\alpha' \xrightarrow{v} \varepsilon \quad \text{and} \quad \beta' \xrightarrow{v} \gamma\right) \qquad \text{or} \qquad \left(\alpha' \xrightarrow{v} \gamma \quad \text{and} \quad \beta' \xrightarrow{v} \varepsilon\right)$$

such that $|v| = \text{min-n}(\alpha', \beta')$ and $\delta \sim \gamma\delta$. As $\text{max-n}(V_N) \leq \text{max-n}(Im(P_f)) + 1$, we have that

$$a_{\alpha,\beta} - 1 \leq \text{min-n}(\alpha, \beta) + |V_N| \cdot \text{max-n}(Im(P_f)).$$

Furthermore,

$$\begin{aligned} n_f(\alpha') &\leq n(\alpha) + |u| \cdot (\text{max-n}_f(Im(P)) - 1) \\ n_f(\beta') &\leq n(\beta) + |u| \cdot (\text{max-n}_f(Im(P)) - 1) \\ n_f(\gamma) &\leq \text{max-n}_f(\alpha', \beta') + |v| \cdot (\text{max-n}_f(Im(P)) - 1) \end{aligned}$$

So

$$\begin{aligned}
n_f(\gamma) &\leq \text{max-n}(\alpha,\beta) + (a_{\alpha,\beta}-1)\cdot(\text{max-n}(\text{Im}(P_f))-1) \\
&+ \Big(\text{min-n}(\alpha,\beta) + (a_{\alpha,\beta}-1)\cdot(\text{max-n}(\text{Im}(P_f))-1)\Big)\cdot \\
&\qquad\qquad \Big(\text{max-n}_f(\text{Im}(P))-1\Big) \\
&\leq \Big(\text{max-n}(\alpha,\beta) + (a_{\alpha,\beta}-1)\cdot(\text{max-n}(\text{Im}(P_f))-1)\Big)\cdot \\
&\qquad\qquad \text{max-n}_f(\text{Im}(P)) \\
&\leq \Big(\text{max-n}(\alpha,\beta)\cdot\text{max-n}(\text{Im}(P_f)) + \\
&\quad |V_N|\cdot\text{max-n}(\text{Im}(P_f))\cdot(\text{max-n}(\text{Im}(P_f))-1)\Big)\cdot\text{max-n}_f(\text{Im}(P)) \\
&< \Big(\text{max-n}(\alpha,\beta) + |V_N|\Big)\cdot\text{max-n}(\text{Im}(P_f))\cdot(\text{max-n}_f(\text{Im}(P)))^2 \\
&= b_{\text{max-n}(\alpha,\beta)}.
\end{aligned}$$

□

We then define the integer $c = b_{\text{max-n}(V_N)\cdot(1+\text{max-n}_f(\text{Im}(P)))}$ to obtain a sequence of decreasing repetitive states.

Lemma 62 *Let $\delta \sim \gamma\delta$ with $0 < n(\gamma) < c$ and max-n$(V_N) < n_f(\delta)$. Then there exist $\widehat{\gamma}$, $\widehat{\delta}$ and ρ such that $\widehat{\delta} \sim \widehat{\gamma}\widehat{\delta}$ with $0 < n(\widehat{\gamma}) < c$ and $\delta \sim \rho\widehat{\delta}$ with $n_f(\delta) = n_f(\rho\widehat{\delta})$ and $0 < n(\rho) < 2$ max-n(V_N).*

Proof As $0 < n(\gamma) < c$, we have that $\gamma = A\gamma''$ with $A \in V_N$. Let $A \xrightarrow{u} \varepsilon$ with $|u| = n(A)$. Then $\delta = \delta'\delta'' \xrightarrow{u} \mu\delta''$ such that $\mu\delta'' \sim \gamma''\delta$ where $|\delta'|$ is minimal. As $n_f(\delta) > \text{max-n}(V_N) \geq |u|$, we have that $\delta'' \neq \varepsilon$ and $\delta' \in V_N^*$. By the mimimality of $|\delta'|$, we have that $\delta' \xrightarrow{u'} B \xrightarrow{u''} \mu$ with $B = \delta'(|\delta'|)$, $u'u'' = u$ and $u'' \neq \varepsilon$. Hence $n(\delta') \leq |u'| + n(B) < 2\cdot\text{max-n}(V_N)$. Furthermore, $n_f(\mu) \leq n(B) + |u''|\cdot(\text{max-n}_f(\text{Im}(P))+1) \leq \text{max-n}(V_N)\cdot\text{max-n}_f(\text{Im}(P))$. We have that $\delta'\delta'' = \delta \sim A\gamma''\delta \sim A\mu\delta''$.

Suppose that $\delta' \sim A\mu$. In particular, $\mu \in V_N^*$, and it suffices to take $\widehat{\gamma} = \gamma''A$, $\widehat{\delta} = \mu\delta''$ and $\rho = A$:

- $\widehat{\delta} = \mu\delta'' \sim \gamma''\delta = \gamma''\delta'\delta'' \sim \gamma''A\mu\delta'' = \widehat{\gamma}\widehat{\delta}$.
- $0 < n(\widehat{\gamma}) = n(\gamma''A) = n(A\gamma'') = n(\gamma) < c$.
- $\delta = \delta'\delta'' \sim A\mu\delta'' = \rho\widehat{\delta}$.
- $n_f(\delta) = n_f(\delta'\delta'') = n(\delta') + n_f(\delta'')$
  $= n(A) + n(\mu) + n_f(\delta'') = n_f(A\mu\delta'') = n_f(\rho\widehat{\delta})$.
- $0 < n(\rho) = n(A) \leq \text{max-n}(V_N) < 2\cdot\text{max-n}(V_N)$.

Assume then that $\delta' \not\sim A\mu$. Then we have that $\delta'\delta'' \sim A\mu\delta''$ with $n(\delta') < 2\cdot\text{max-n}(V_N)$ and $n(A\mu) \leq \text{max-n}(V_N)\cdot(1+\text{max-n}_f(\text{Im}(P)))$.

- If $\mu \in V_N^*$, then by Lemma 61, there is $\widehat{\gamma}$ such that $\delta'' \sim \widehat{\gamma}\delta''$ and

$$n_f(\widehat{\gamma}) \;<\; b_{\max(2\cdot\text{max-n}(V_N),\text{max-n}(V_N)\cdot(1+\text{max-n}_f(\text{Im}(P))))} \;=\; c.$$

- If $n(\mu) = \infty$, then $\delta'\delta'' \sim A\mu$. Let $\delta' \xrightarrow{\nu} \varepsilon$ with $|\nu| = n(\delta')$. There is $A\mu \xrightarrow{\nu} \widehat{\gamma}$ with $\delta'' \sim \widehat{\gamma} \sim \widehat{\gamma}\delta''$. Then $n_f(\widehat{\gamma}) \leq n_f(A\mu) + |\nu| \cdot (\text{max-n}_f(\text{Im}(P)) - 1)$. Thus $n_f(\widehat{\gamma}) < \text{max-n}(V_N) \cdot (1 + \text{max-n}_f(\text{Im}(P))) + 2 \cdot \text{max-n}(V_N) \cdot (\text{max-n}_f(\text{Im}(P)) - 1) < c$. So it suffices to take $\widehat{\delta} = \gamma''$ and $\rho = \gamma'$.

□

We bound the finite norm of repetitive states with the following integer: $d = 2 \cdot \text{max-n}(V_N) \cdot |V_N|^c$.

Lemma 63 *Let $\delta \sim \gamma\delta$ with $n_f(\gamma) < c$. Then there is $\widehat{\delta} \sim \delta$ with $n_f(\widehat{\delta}) < d$.*

Proof If $n(\gamma) = \infty$ then it suffices to take $\widehat{\delta} = \delta$ (since $c < d$). Let $\delta \sim \gamma\delta$ with $n(\gamma) < c$. We may assume that $n_f(\delta)$ is minimal. By repeated applications of Lemma 62, there is a finite sequence $(\gamma_i, \delta_i, \rho_i)$ with $1 \leq i \leq n$ such that

- $\gamma_1 = \gamma$ and $\delta_1 = \delta$;
- $n_f(\delta_n) \leq \text{max-n}(V_N)$;
- for each $1 \leq i \leq n$: $\delta_i \sim \gamma_i\delta_i$ and $0 < n(\rho_i) < 2 \cdot \text{max-n}(V_N)$; and
- for each $1 \leq i < n$: $\delta_i \sim \rho_{i+1}\delta_{i+1}$ and $n_f(\delta_i) = n_f(\rho_{i+1}\delta_{i+1})$.

Assume that $\gamma_i = \gamma_j$ for $i < j$. Then $\delta_i \sim \delta_j$ with $n_f(\delta_j) < n_f(\delta_i)$. If $\widehat{\delta} = \rho_2 \ldots \rho_i\delta_j$ then $\widehat{\delta} \sim \rho_2 \ldots \rho_i\delta_i \sim \delta_1 = \delta$, but $n_f(\delta) = n_f(\rho_2 \ldots \rho_i\delta_i) > n_f(\widehat{\delta})$. This contradiction to the minimality of δ implies that the γ_i are pairwise distincts. Hence

$$\begin{aligned} n &\leq |\{\gamma \in V_N^* \,:\, n(\gamma) < c\}| \\ &\leq 1 + |V_N| + \ldots + |V_N|^{c-1} \\ &= (|V_N|^c - 1)/(|V_N| - 1) \;<\; |V_N|^c. \end{aligned}$$

Finally

$$\begin{aligned} n_f(\delta) &= n_f(\rho_2 \ldots \rho_n\delta_n) \;<\; 2(n-1) \cdot \text{max-n}(V_N) + \text{max-n}(V_N) \\ &< 2n \,\text{max-n}(V_N) \;<\; 2\,\text{max-n}(V_N) \cdot |V_N|^c \;=\; d. \end{aligned}$$

□

The integer $e = d + 1 + \text{max-n}(V_N) \cdot \text{max-n}(\text{Im}(P_f))$ is then a suitable bound.

Theorem 64 *The relation R_e is complete: $\overset{R_e \cap \sim}{\equiv} = \sim$.*

Proof We have $\overset{R_e \cap \sim}{\equiv} \subseteq \tilde{\equiv} = \sim$. To prove the reverse inclusion and using R_0 it suffices to show that for every $\alpha, \beta \in V_N^* \cup V_N^* V_u$, if $\alpha \sim \beta$ then $\alpha \overset{R_e \cap \sim}{\equiv} \beta$. We may also assume that $\text{Im}(P) \subseteq V_N^* \cup V_N^* V_u$. In the sequel, we let $\equiv = \overset{R_e \cap \sim}{\equiv}$.

The proof is carried out by induction on $\preceq$ as defined in Definition 33: $(\alpha,\beta) \preceq (\gamma,\delta)$ iff $\max\text{-}n_f(\alpha,\beta) \leq \max n_f(\gamma,\delta)$.

Suppose first that $\max\text{-}n_f(\alpha,\beta) = 0$, that is, $\alpha,\beta \in V_u \cup \{\varepsilon\}$. As $\alpha \sim \beta$, either $\alpha = \beta = \varepsilon$, in which case $\alpha \equiv \beta$; or $\alpha,\beta \in V_u$, in which case $\alpha(R_e \cap \sim)\beta$.

Assume then that $\max\text{-}n_f(\alpha,\beta) > 0$. First, let us verify the following property of normed sequences $\lambda,\mu \in V_N^*$:

$$(\star) \quad \text{If} \quad \mu \sim \lambda \quad \text{and} \quad n_f(\mu) < e \quad \text{and} \quad (\lambda,\lambda) \prec (\alpha,\beta) \quad \text{then} \quad \mu \equiv \lambda.$$

Either $n(\lambda) < \infty$ in which case $(\mu,\lambda) \prec (\alpha,\beta)$ and by the induction hypothesis $\mu \equiv \lambda$; or $n(\lambda) = \infty$ in which case either $n_f(\lambda) < e$ so $\mu(R_e \cap \sim)\lambda$, or $n_f(\lambda) \geq e$ in which case $(\mu,\lambda) \prec (\lambda,\lambda) \prec (\alpha,\beta)$, and by the induction hypothesis $\mu \equiv \lambda$.

Now let $A\alpha' = \alpha$ and $B\beta' = \beta$. By symmetry, we may assume that $n(B) \leq n(A)$, so $B \in V_N$. Let $B \xrightarrow{u} \varepsilon$ with $|u| = n(B)$. Then $A \xrightarrow{u} \gamma$ such that $\gamma\alpha' \sim \beta'$. In particular,

$$\begin{aligned} n_f(\gamma) &\leq |u| \cdot (\max\text{-}n_f(Im(P)) - 1) + 1 \\ &\leq \max\text{-}n(V_N) \cdot (\max\text{-}n_f(Im(P)) - 1) + 1 < e. \end{aligned}$$

Furthermore, $A\alpha' \sim B\gamma\alpha'$.

Suppose $n(\gamma) = \infty$. Then $\gamma \sim \beta'$, and by Property $(\star)$, $\gamma \equiv \beta'$. Furthermore $A\alpha' \sim B\gamma$ and $n_f(B\gamma) \leq \max\text{-}n(V_N) \cdot \max\text{-}n_f(Im(P)) + 1 < e$.

- If $A \in V_u$ then $\alpha' = \varepsilon$ and hence $\alpha = A(R_e \cap \sim)B\gamma \equiv B\beta' = \beta$.
- If $A \in V_N$ then $A \xrightarrow{v} \varepsilon$ such that $|v| = n(A)$. Hence $B\gamma \xrightarrow{v} \widehat{\alpha}$ such that $\alpha' \sim \widehat{\alpha}$. In particular, $n_f(\widehat{\alpha}) \leq \max\text{-}n(V_N) \cdot (\max\text{-}n_f(Im(P)) - 1) + n_f(B\gamma) \leq 2 \cdot \max\text{-}n(V_N) \cdot \max\text{-}n_f(Im(P)) < e$ and by Property $(\star)$, $\widehat{\alpha} \equiv \alpha'$. Furthermore, $A\widehat{\alpha} \sim B\gamma$ and $n_f(A\widehat{\alpha}) \leq \max\text{-}n(V_N) \cdot (2 \cdot \max\text{-}n_f(Im(P)) + 1) < e$. Hence $A\widehat{\alpha}(R_e \cap \sim)B\gamma$. Finally, $\alpha = A\alpha' \equiv A\widehat{\alpha} \equiv B\gamma \equiv B\beta' = \beta$.

Assume then that $n(\gamma) < \infty$. Then $A \in V_N$, and since $n(\gamma\alpha') = n(\beta')$, by induction, $\gamma\alpha' \equiv \beta'$. Furthermore, $n(\gamma) \leq |u| \cdot (\max\text{-}n(Im(P_f)) - 1) + 1 \leq \max\text{-}n(V_N) \cdot (\max\text{-}n(Im(P_f)) - 1) + 1$.

- If $A \sim B\gamma$ then $A(R_e \cap \sim)B\gamma$. Hence $\alpha = A\alpha' \equiv B\gamma\alpha' \equiv B\beta' = \beta$.
- If $A \not\sim B\gamma$ then $A\alpha' \sim B\gamma\alpha'$. By Lemma 61, $\alpha' \sim \delta\alpha'$ for some δ with $n_f(\delta) < b_{\max\text{-}n(A,B\gamma)} < c$. By Lemma 63, $\alpha' \sim \widehat{\delta}$ for some $\widehat{\delta}$ with $n_f(\widehat{\delta}) < d$. By Property $(\star)$, $\alpha' \equiv \widehat{\delta}$. Furthermore, $A\widehat{\delta}(R_e \cap \sim)B\gamma\widehat{\delta}$, since $A\widehat{\delta} \sim B\gamma\widehat{\delta}$ with $n_f(A\widehat{\delta}) < e$ and

$$\begin{aligned} n_f(B\gamma\widehat{\delta}) &< \max\text{-}n(V_N) + n(\gamma) + d \\ &\leq \max\text{-}n(V_N) \cdot \max\text{-}n(Im(P_f)) + 1 + d \;=\; e. \end{aligned}$$

  Finally, $\alpha = A\alpha' \equiv A\widehat{\delta} \equiv B\gamma\widehat{\delta} \equiv B\gamma\alpha' \equiv B\beta' = \beta$.

□

Although we have a direct decision procedure to decide bisimilarity for any BPA system, the complexity is not polynomial: constant e is exponential in the

size of the system. A close analysis would demonstrates that the complexity of the underlying algorithm is in fact doubly exponential in the size of the BPA system.

2.4 Undecidability Results for MSA

In this section we outline Jančar's technique [84] for demonstrating undecidability results for bisimilarity through mimicking Minsky machines in a weak fashion but faithfully enough to capture the essence of the halting problem. Jančar first employed his ideas to demonstrate the undecidability of bisimulation equivalence over the class of Petri nets. Hirshfeld [66] modified the argument to demonstrate the undecidability of trace equivalence over BPP. Jančar and Moller [89] then used the technique to demonstrate the undecidability of regularity checking of Petri nets with respect to both simulation and trace equivalence.

In the following we generalize Jančar's result by demonstrating the undecidability of (normed) MSA. Jančar's technique applies ideally in this case, as MSA offer precisely the ingredients present in Petri nets which are needed to mimic Minsky machines.

Minsky machines [117] 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

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

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 $\mathtt{X}_n$. We recall now the fact that the halting problem for Minsky machines is undecidable: there is no algorithm which decides whether or not a given Minsky machine halts.

A Minsky machine as presented above gives rise to the following MSA.

- The input alphabet is $\Sigma = \{\mathtt{i}, \mathtt{d}, z, \omega\}$.
- The control states are $Q = \{\mathtt{p}_0, \mathtt{p}_1, \ldots, \mathtt{p}_{n-1}, \mathtt{p}_n, \mathtt{q}_0, \mathtt{q}_1, \ldots, \mathtt{q}_{n-1}, \mathtt{q}_n\}$.

- The stack alphabet is $\Gamma = \{Z, 0, 1\}$.
- For each machine instruction

$$X_\ell : c_b := c_b{+}1;\ \texttt{goto}\ X_j$$

  we have the MSA rules

$$p_\ell Z \xrightarrow{i} p_j b Z \qquad \text{and} \qquad q_\ell Z \xrightarrow{i} q_j b Z.$$

- For each machine instruction

$$\begin{aligned} X_\ell : \ & \texttt{if}\ c_b = 0\ \texttt{then goto}\ X_j \\ & \texttt{else}\ c_b := c_b{-}1;\ \texttt{goto}\ X_k \end{aligned}$$

  we have the MSA rules

$$\begin{array}{lll} p_\ell b \xrightarrow{d} p_k & p_\ell Z \xrightarrow{z} p_j Z & p_\ell b \xrightarrow{z} q_j b \\ q_\ell b \xrightarrow{d} q_k & q_\ell Z \xrightarrow{z} q_j Z & q_\ell b \xrightarrow{z} p_j b \end{array}$$

- We have the one final MSA rule

$$p_n Z \xrightarrow{\omega} p_n$$

The two states $p_0 Z$ and $q_0 Z$ of this MSA each mimic the machine M in the following sense.

- When M is at the command labelled X_ℓ with the values x and y in its counters, this is reflected by the MSA being in state $p_\ell 0^x 1^y Z$ (or $q_\ell 0^x 1^y Z$).
- If this command is an increment, then the MSA has only one transition available from the control state p_ℓ (q_ℓ), which is labelled by i (for 'increment'), resulting in the MSA state reflecting the state of the machine upon executing the increment command.
- If this command is a successful test for zero (that is, the relevant counter has the value 0), then the MSA has only one transition available from the control state p_ℓ (q_ℓ), which is labelled z (for 'zero'), again resulting in the MSA state reflecting the state of the machine upon executing the test for zero command.
- If this command is a decrement (that is, a failed test for zero), then the MSA in control state p_ℓ (q_ℓ) has three possible transitions, exactly one of which is labelled d (for 'decrement') which would once again result in the MSA state reflecting the state of the machine upon executing the decrement command.

- In this last instance, the MSA has the option to disregard the existence of a relevant counter symbol in the stack and behave as if the program counter was zero. This reflects the weakness of Petri nets (and hence MSA) in their inability to test for zero (a weakness which works in their favour with respect to several important positive decidability results such as the reachability problem [106]). In this case, the MSA in control state p_ℓ (q_ℓ) may make a z transition in either of two ways: either by "honestly" cheating using the rule $p_\ell Z \xrightarrow{z} p_j Z$ ($q_\ell Z \xrightarrow{z} q_j Z$), or by "knowingly" cheating using the rule $p_\ell b \xrightarrow{z} q_j b$ ($q_\ell b \xrightarrow{z} p_j b$) thus moving the control state over into the domain of the other MSA mimicking M.

Fact 65 $p_0 Z \sim q_0 Z$ *iff the Minsky machine* M *does not halt.*

Proof If M halts, then a winning strategy for player I in the bisimulation game would be to mimic the behaviour of M in either of the two MSA states. Player II's only option in response would be to do the same with the other MSA state. Upon termination, the game states will be $p_n 0^x 1^y Z$ and $q_n 0^x 1^y Z$ for some values x and y. Player I may then make the transition $p_n 0^x 1^y Z \xrightarrow{\omega} p_n 0^x 1^y$ which cannot be answered by player II from the state $q_n 0^x 1^y Z$. Hence $p_0 Z$ and $q_0 Z$ cannot be bisimilar.

If M fails to halt, then a winning strategy for player II would be to mimic player I's moves for as long as player I mimics M, and to cheat knowingly or honestly, respectively, in the instance that player I cheats honestly or knowingly, respectively, so as to arrive at the situation where the two states are identical; from here player II can copy every move of player I verbatim. Hence $p_0 Z$ and $q_0 Z$ must be bisimilar. □

We thus have undecidability of bisimulation equivalence over a very restricted class of Petri nets: those with only two unbounded places and a minimal degree of nondeterminism. Note that this nondeterminism is essential: Jančar [84] shows that bisimulation equivalence is decidable between two Petri nets when one of them is deterministic up to bisimilarity.

The above MSA can be made into a normed rewrite transition system by adding a new input symbol n along with the following MSA rules (one for each $\ell = 0 \ldots n$ and each $X = Z, 0, 1$).

$$p_\ell X \xrightarrow{n} p_\ell \qquad q_\ell X \xrightarrow{n} q_\ell$$
$$p_\ell X \xrightarrow{n} q_\ell \qquad q_\ell X \xrightarrow{n} p_\ell$$

These moves allow the MSA to exhaust its stack at any point during its execution, and continues to allow player II to produce a pair of identical states if player I elects to take one of these non-M-mimicking transitions. The same argument can then be made to show that $p_0 Z$ and $q_0 Z$ are bisimilar exactly when the Minsky machine M does not halt.

Theorem 66 *Bisimilarity is undecidable over the class of normed MSA.*

This result contrasts with that of Stirling [137] regarding the decidability of bisimilarity over the class of normed PDA.

	$\sim$	$\sim$ FSA	$\sim$-regularity
BPA	2EXPTIME [23] PTIME in the normed case [70]	PTIME [99]	2EXPTIME [24]
PDA	decidable [134] PSPACE-hard [108]	EXPTIME [87] PSPACE-hard [108]	PSPACE-hard [108]
BPP	decidable [36] co-NP-hard [107] PTIME in the normed case [71]	PSPACE [87]	decidable [55] co-NP-hard [107]
PA	co-NP-hard [107] decidable in the normed case [68]	decidable [87]	co-NP-hard [107]
PN	undecidable [85]	decidable [55] EXPSPACE-hard [108]	decidable [55] EXPSPACE-hard [108]

Figure 2: Results for bisimilarity.

2.5 Summary of Results

The tables in Figures 2 and 3 summarise a variety of decidability and algorithmic results for bisimulation problems over several classes of infinite-state systems, many of which being as described above, as well as give pointers to where to find these results in the literature. The first column of the first table lists results regarding bisimilarity as discussed in this chapter; the second column lists results regarding the problem of comparing a system to a finite-state system; and the third column lists results regarding the *regularity problem*, determining if the system is equivalent to some unspecified finite-state system. The second table presents the same collection of results, but this time in relation to *weak bisimulation equivalence*. To define weak bisimilarity, we assume that there is some label τ which represents an unobservable transition label, and we define a new transition relation $\Longrightarrow$ by: $\stackrel{\tau}{\Longrightarrow} = \stackrel{\tau}{\longrightarrow}^*$, and for $a \neq \tau$, $\stackrel{a}{\Longrightarrow} = \stackrel{\tau}{\longrightarrow}^* \cdot \stackrel{a}{\longrightarrow} \cdot \stackrel{\tau}{\longrightarrow}^*$. Then the definition of weak bisimilarity is the same as for bisimilarity except using this new transition relation $\Longrightarrow$ in place of the original $\longrightarrow$ relation.

One-Counter Automata and One-Counter Nets

The discussion of equivalence checking in this chapter only touches the surface of this vast topic, and many related studies have been carried out which consider other classes of systems and other equivalences. In particular, there has recently been extensive work done on one-counter automata and one-counter nets, for both bisimulation equivalence and *simulation preorder/equivalence*.

A *simulation relation* is defined as a "one-sided" bisimulation relation, in

	$\approx$	$\approx$ FSA	$\approx$-regularity
BPA	PSPACE-hard [139]	PTIME [99]	?
PDA	PSPACE-hard [139]	EXPTIME [87] PSPACE-hard [108]	PSPACE-hard [108]
BPP	NP-hard [139] Π_2^P-hard [107]	PSPACE [87]	Π_2^P-hard [107]
PA	PSPACE-hard [139]	decidable [87]	Π_2^P-hard [107]
PN	undecidable [85]	undecidable [55]	undecidable [55]

Figure 3: Results for weak bisimilarity.

the sense that we omit the second clause in Definition 9, and in the third clause we only demand that β be a final state whenever α is, but not vice versa. We then say that α is *simulated* by β if the pair (α, β) is contained in some simulation relation; that is, the simulation preorder is defined to be the union of all simulation relations (and hence the largest simulation relation). Finally, two states are *simulation equivalent* if they simulate each other.

The first relevant result is that of Jančar [86], demonstrating that bisimulation equivalence, as well as the regularity problem with respect to bisimilarity, are decidable for one-counter automata. Although this is subsumed by the later result of Sénizergues [132], the technique used by Jančar in his proof of the special case is elegant, employing novel ideas involving representing bisimilarity as a colouring of the plane and demonstrating the existence of a periodicity which can be detected.

Abdulla and Čerāns [1] recently outlined an extensive and technically-challenging proof of the decidability of the simulation preorder for one-counter nets. This result was subsequently given a concise and intuitive proof by Jančar and Moller [90], again using colourings of the plane.

For the slightly wider class of one-counter automata, Jančar, Moller and Sawa [92] demonstrate the undecidability of simulation equivalence, and hence also simulation preorder; for the equivalence problem, we can even assume that one of the automata is deterministic, and for the preorder, we can assume that both are deterministic. Contrasting with these results are, firstly, the decidability of the equivalence problem for deterministic one-counter automata, a result which can be extracted from the early results of Valiant and Paterson [143]; and secondly, the recent result of Jančar, Kučera and Moller [88] that equivalence is decidable between a deterministic one-counter automaton (in fact, a deterministic PDA even) and a one-counter net.

Finally, it is also demonstrated in [88] that the regularity problem for one-counter nets with respect to simulation equivalence is decidable. (The corresponding problem is undecidable for both Petri nets [89] and PA [98].) This

proof is carried out by presenting an elegant general reduction from simulation problems over one-counter nets to bisimulation problems over one-counter automata, and then exploiting the original result of Jančar [86].

3 The Model Checking Problem

In this section we survey the state-of-the-art in model-checking infinite-state systems. Again, we concentrate exclusively on discrete infinite-state systems and neglect, e.g., dense infinite-state structures related to timed or hybrid systems, which are covered, for instance, in [3] and [2]. Due to the broad scope of existing model checking algorithms, we have chosen to sketch only the main ideas behind the relevant results complemented by appropriate references: for decidable model checking problems we briefly present the developed algorithms together with their complexity; for undecidable problems, we outline the undecidability proof. This style of presentation is intended to provide readers interested in more technical details with access to the original papers, and ensures at the same time that the survey remains readable for those who wish only to glance over the research done in this area.

The remainder of this section is organized as follows. We briefly introduce in Section 3.1 the temporal logics we consider together with their branching and linear time classifications; this introduction can be deepened by consulting chapter 1..4 on "Modal Logics for Processes" in this handbook [17]. We then proceed by presenting decidability and complexity results about model checking for various classes of infinite-state systems, first for branching time logics in Section 3.2, and subsequently for linear time logics in Section 3.3. The presentation is organized according to the two central parameters of the model checking problem:

- the size of the representation of the transition system; and
- the size of the temporal formula.

This allows us to distinguish three different complexities:

- the general case;
- the case where the formula is fixed; and
- the case where the system is fixed.

As in practice the size of the formula is usually very small compared to the size of the system representation, we focus on the first two cases here and neglect the last one in our complexity considerations.

3.1 Temporal Logics

A successful approach to the verification of programs relies on a suitable formalism for specifying central aspects of the intended system behaviour. In the case of *sequential* programs where the input/output behaviour together with termination plays a predominant role such formalisms are traditionally based on state transformer semantics. In fact, sequential programs are typically verified by considering their (partial) correctness, specified in terms of pre- and postconditions [58].

Reactive systems, on the other hand, are typically *nonterminating*, as they maintain an ongoing interaction with the environment. Hence, verification methods which rely intrinsically on the existence of a final state are, usually, not applicable and must be replaced by radically different approaches (see [100] for a survey). Pnueli [127] was the first to recognize the need for formalisms which support reasoning about nonterminating behaviour. He proposed the use of *temporal logic* as a language for the specification of concurrent program properties. Temporal logics are tailored to expressing many important correctness properties characteristic for reactive systems, in particular, liveness properties which assert that something will eventually happen.

Since Pnueli's landmark paper a plethora of systems of temporal logic have been investigated concerning their expressiveness and suitability for verification purposes. The majority and better known of these logics belong to the class of point-based, future-tense, and propositional logics. In this survey we will also focus on this class, while elaborating on the differences between branching-time logics and linear-time logics. The reader interested in technical details beyond this exposition is adviced to consult [120] and chapter 1.4 on "Modal Logics for Processes" in this handbook [17].

Within their respective spectra, branching and linear-time logics can be classified according to their expressive power. Figure 4 shows the respective hierarchies of temporal logics we will consider in the following. In this figure an arrow $L_1 \rightarrow L_2$ indicates that the logic L_2 is strictly more expressive than the logic L_1.

Due to their difference in expressive power the various logics have also different importance for verification purposes. The weakest logics, Hennessy-Milner logic and weak linear-time logic, can only express properties about a finite prefix of a system, and play consequently only a minor role in themselves. They constitute, however, the basis for the more expressive logics and possess some interesting theoretical properties. Most importantly, for finite-branching processes, i.e., for processes where each state admits only finitely many transitions, it is known that two processes are bisimulation equivalent iff they satisfy the same set of Hennessy-Milner logic formulae [65]. To increase the expressiveness of these basic logics, which require infinite sets of formulae for characterizing infinite behaviour, the addition of the "until" operator has been proposed which can express that a property ϕ should hold *until* a second property ψ holds. On the branching-time side this leads to Computation Tree Logic (CTL) [42], while on the linear-time side this yields Linear Temporal Logic (LTL) [127]. Since both logics arise quite naturally and can express practically relevant properties they are widely used in tools for automatic verification. But even the logics EF and EG, complementary fragments of CTL, have stirred up some interest. They both have been successfully used to clarify the borderline between decidability and undecidability, as well as to establish lower complexity bounds for model checking various classes of infinite-state systems. The logic UB is simply the smallest common generalization of EF and EG.

Even more expressive temporal logics are obtained by adding least and greatest fixpoint operators to the basic logics. This extension results in the

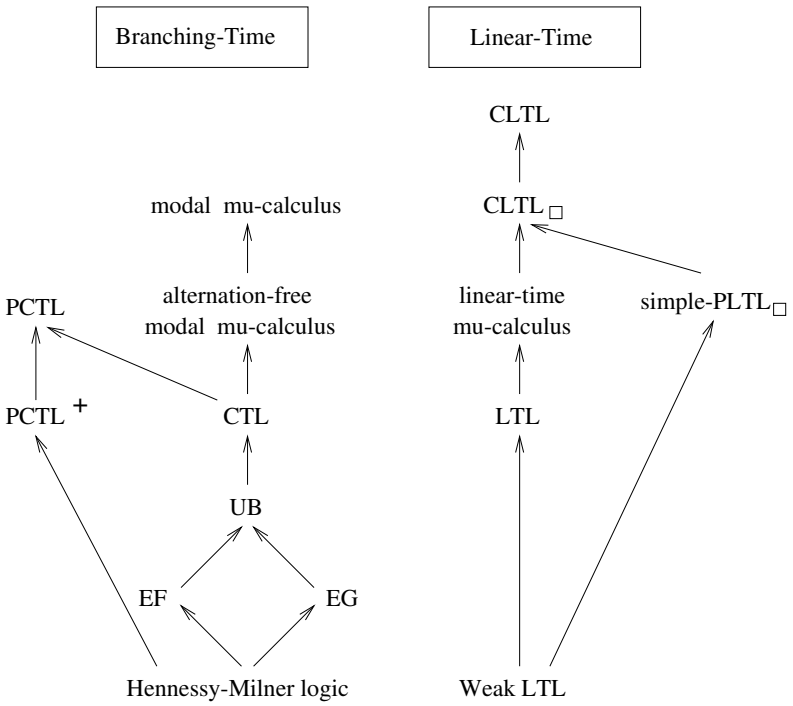


Figure 4: Branching and linear time logics.

branching-time modal μ-calculus and the linear-time μ-calculus. Both fixpoint logics play a central role in the theory of model checking due to their syntactic minimality and their pleasing mathematical properties. Accordingly, we will present both logics together with their respective model checking algorithms in greater detail.

All the logics mentioned so far have in common that they may express only *regular properties*. This means that in the branching-time setting the set of states, respectively in the linear-time setting the set of traces, satisfying a given property form regular sets. Whereas the meaning of regularity is clear for traces, regular sets of states need to be explained: a state in a transition system characterizes a tree, and for trees regularity is well-defined [128]. There have, however, been extensions proposed which incorporate nonregular features like counting. The most prominent examples are Presburger Computation Tree Logic (PCTL) [10] which combines CTL with Presburger arithmetic, and Constrained Linear Temporal Logic (CLTL) [13], which enhances LTL by Presburger arithmetic and constraints expressed by finite-state automata. Despite the fact that both logics in their general form are undecidable already for finite-state systems, decidability can be regained for nontrivial fragments.

For the following formal introduction of characteristic temporal logics let $\mathcal{T} = (\mathcal{S}, \Sigma, \rightarrow)$ be a labelled transition system without start and final states, Σ^*, resp. Σ^ω, denote the set of finite, resp. infinite words over Σ, and $\Sigma^\infty =_{\mathrm{df}} \Sigma^* \cup \Sigma^\omega$.

Moreover, given a word $\sigma = a_1 a_2 \ldots \in \Sigma^\infty$, $\sigma(2)$ will denote the first action of σ, i.e., a_1, and σ^1 the word $a_2 a_3 \ldots$.

A *path* $\pi = s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \ldots$ in $\mathcal{T}$ is a finite or infinite transition sequence such that $s_i \xrightarrow{a_i} s_{i+1}$, for all i, is in the transition relation. By $\pi(i)$ we denote its i-th state s_i, whereas π^i denotes the part of π starting at s_i. Moreover, we denote by $\pi(\vdash)$ its first state, and, if π is finite, by $\pi(\dashv)$ its last state, respectively. A *run* is a maximal path, i.e., a path that is either infinite or terminates in a state without successors. Furthermore, we denote by $\mathtt{paths}(s)$ the set of paths starting in s, while $\mathtt{runs}(s)$ will denote the set of runs starting in s. The set of finite paths from s is written $\mathtt{paths}^*(s)$. Finally, we write $\mathtt{prefs}(\pi)$ for the set of finite prefixes of the path π.

3.1.1 Branching-Time Logics

The main characteristic of branching-time logics is that they model the potential of possible futures in a tree-like structure: each moment in time has at least one, but may have even infinitely many possible successor moments. Together with the temporal order on moments in time, this structure therefore defines infinite trees, which, in a natural way, correspond to computation trees of concurrent processes. Consequently, formulae of a branching-time temporal logic can be conveniently used to loosely specify the behaviour of concurrent systems according to their structural operational semantics. Moreover, as many process algebras assume a set Σ of observable actions, modal operators of branching-time logics often quantify over action-labelled transitions allowing to specify which property should hold after a specific action has been observed.

Hennessy-Milner Logic (HML) The weakest branching-time logic is *Hennessy-Milner logic* [65], built out of "true", negation, conjunction, and the relativized existential next operator $\langle a \rangle$. Formally, formulae have the following syntax:

$$\phi ::= \mathtt{tt} \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \langle a \rangle \phi$$

where a is an action of Σ. As is characteristic for a branching-time logic, the semantics $[\![\phi]\!]^{\mathcal{T}}$ of a formula ϕ is defined with respect to a labelled transition system $\mathcal{T}$, and denotes the subset of the set of states $\mathcal{S}$ for which the formula holds. In particular, for formulae of Hennessy-Milner logic the semantics is defined inductively as follows:

$$\begin{aligned}
[\![\mathtt{tt}]\!]^{\mathcal{T}} &=_{df} \mathcal{S} \\
[\![\neg\phi]\!]^{\mathcal{T}} &=_{df} \mathcal{S} - [\![\phi]\!]^{\mathcal{T}} \\
[\![\phi_1 \wedge \phi_2]\!]^{\mathcal{T}} &=_{df} [\![\phi_1]\!]^{\mathcal{T}} \cap [\![\phi_2]\!]^{\mathcal{T}} \\
[\![\langle a \rangle \phi]\!]^{\mathcal{T}} &=_{df} \{\, s \in \mathcal{S} \mid \exists\, s' \in \mathcal{S}.\ s \xrightarrow{a} s' \text{ and } s' \in [\![\phi]\!]^{\mathcal{T}} \,\}
\end{aligned}$$

Instead of $s \in [\![\phi]\!]^{\mathcal{T}}$ we also write $s \models \phi$. To ease the notation further, the dual notions *false*, disjunction and universal next operator are defined using

the standard abbreviations

$$\begin{aligned} \mathtt{ff} &=_{\mathrm{df}} \neg\mathtt{tt} \\ \phi_1 \vee \phi_2 &=_{\mathrm{df}} \neg(\neg\phi_1 \wedge \neg\phi_2) \\ [a]\phi &=_{\mathrm{df}} \neg\langle a\rangle\neg\phi \end{aligned}$$

Under the customary assumption that atomic propositions are closed under negation, these abbreviations allow us to transform every formula of Hennessy-Milner logic into *positive normal form*, i.e., an equivalent formula without negation, by driving negation inwards.

From a conceptual point of view Hennessy-Milner logic is easy to deal with as validity of a formula at a state s may readily be checked for any process whose computation tree is finite-branching, i.e., has only a finite number of alternatives for each observable action. It simply suffices to investigate the computation tree of the labelled transition graph under consideration up to depth $|\phi|$, where $|\phi|$ denotes the size of ϕ defined in the usual way by counting the number of operators.

The Logics EF, EG and UB The logic EF [50] is obtained from Hennessy-Milner logic by adding the operator $\mathsf{EF}\phi$, meaning "there exists a path such that eventually ϕ holds". This addition increases the expressive power of the logic significantly, as it becomes possible to reason about paths of arbitrary length. In fact, it already suffices to express the practically relevant *reachability* properties and the practically even more important dual *invariance* or *safety* properties: $\neg\mathsf{EF}\phi$ expresses that a state satisfying a certain undesirable property ϕ cannot be reached.

The addition of the operator $\mathsf{EG}\phi$ to HML, meaning "there exists a path such that ϕ always holds" yields, symmetrically, the logic EG, which allows to express *inevitability properties*, e.g., $\neg\mathsf{EG}\phi$ means that the considered system is guaranteed to eventually violate property ϕ. The combination of the logics EF and EG yields the obviously even more expressive *Unified System of Branching-Time Logic* (UB) [7].

Formally the semantics of the operators EF and EG are given as follows:

$$\begin{aligned} [\![\mathsf{EF}\phi]\!]^{\mathcal{T}} &=_{\mathrm{df}} \{\, s \in \mathcal{S} \mid \exists\, \pi \in \mathtt{paths}^*(s).\ \pi(\dashv) \in [\![\phi]\!]^{\mathcal{T}} \,\} \\ [\![\mathsf{EG}\phi]\!]^{\mathcal{T}} &=_{\mathrm{df}} \{\, s \in \mathcal{S} \mid \exists\, \pi \in \mathtt{runs}(s).\ \forall\, \pi' \in \mathtt{prefs}(\pi).\ \pi'(\dashv) \in [\![\phi]\!]^{\mathcal{T}} \,\} \end{aligned}$$

Both operators have also a dual operator which in the case of EF is defined as $\mathsf{AG}\phi =_{\mathrm{df}} \neg\mathsf{EF}\neg\phi$, and in the case of EG as $\mathsf{AF}\phi =_{\mathrm{df}} \neg\mathsf{EG}\neg\phi$. Intuitively, they express properties of the form "always in the future ϕ holds" and "eventually in the future ϕ holds".

Finally, a useful complexity measure for formulae in EF, respectively EG, is their *nesting depth* which measures the nesting of EF, respectivelyEG operators.

Computation Tree Logic (CTL) *Computation Tree Logic* (CTL) [42] is one of the earliest proposed branching-time logics. It can be considered as the branching-time counterpart of Linear Temporal Logic (LTL) [127], which was

proposed earlier and which will be introduced in the next section. CTL adds to Hennessy-Milner logic an *until* operator which is either existentially quantified $\mathsf{E}[\phi \,\mathsf{U}\, \psi]$ or universally quantified $\mathsf{A}[\phi \,\mathsf{U}\, \psi]$. The formal semantics of the new operators are as follows.

$$\begin{aligned}
[\![\mathsf{E}[\phi \,\mathsf{U}\, \psi]]\!]^{\mathcal{T}} =_{\mathrm{df}} \{\, s \in \mathcal{S} \mid\ & \exists\, \pi \in \mathtt{paths}^*(s).\ \forall\, \pi' \in \mathtt{prefs}(\pi). \\
& (\pi' \neq \pi \Rightarrow \pi'(\dashv) \in [\![\phi]\!]^{\mathcal{T}}) \wedge \pi(\dashv) \in [\![\psi]\!]^{\mathcal{T}} \,\} \\
[\![\mathsf{A}[\phi \,\mathsf{U}\, \psi]]\!]^{\mathcal{T}} =_{\mathrm{df}} \{\, s \in \mathcal{S} \mid\ & \forall\, \pi \in \mathtt{paths}^*(s).\ \forall\, \pi' \in \mathtt{prefs}(\pi). \\
& (\pi' \neq \pi \Rightarrow \pi'(\dashv) \in [\![\phi]\!]^{\mathcal{T}}) \wedge \pi(\dashv) \in [\![\psi]\!]^{\mathcal{T}} \,\}
\end{aligned}$$

Intuitively, $\mathsf{E}[\phi \,\mathsf{U}\, \psi]$ means that there exists a path on which ϕ holds until, eventually, ψ holds, while $\mathsf{A}[\phi \,\mathsf{U}\, \psi]$ expresses that on all paths ϕ holds until, eventually, ψ holds. The until operators are expressive enough to encode both EF and EG by means of the following formulae.

$$\begin{aligned}
\mathsf{EF}\phi &\equiv \mathsf{E}[\mathtt{tt} \,\mathsf{U}\, \phi] \\
\mathsf{EG}\phi &\equiv \neg\mathsf{A}[\mathtt{tt} \,\mathsf{U}\, \neg\phi]
\end{aligned}$$

As a consequence, CTL subsumes the logic UB.

An example that demonstrates the practical applicability of CTL is the specification of the alternation bit protocol, a well-known communication protocol for exchanging messages between a sender process and a receiver process. The alternation bit protocol could be specified by the CTL formulae

$$\mathsf{AG}(\mathrm{RcvMsg} \Rightarrow \mathsf{A}[\mathrm{RcvMsg} \;\mathsf{U}\; (\neg\mathrm{RcvMsg} \wedge \mathsf{A}[\neg\mathrm{RcvMsg} \;\mathsf{U}\; \mathrm{SndMsg}])])$$

$$\begin{aligned}
&\mathsf{AG}(\mathrm{SndMsg} \wedge \mathrm{Snd0} \Rightarrow \\
&\quad \mathsf{A}[\mathrm{SndMsg} \;\mathsf{U}\; (\neg\mathrm{SndMsg} \wedge \mathsf{A}[\neg\mathrm{SndMsg} \;\mathsf{U}\; (\mathrm{RcvMsg} \wedge \mathrm{Rcv0})])])
\end{aligned}$$

$$\begin{aligned}
&\mathsf{AG}(\mathrm{SndMsg} \wedge \mathrm{Snd1} \Rightarrow \\
&\quad \mathsf{A}[\mathrm{SndMsg} \;\mathsf{U}\; (\neg\mathrm{SndMsg} \wedge \mathsf{A}[\neg\mathrm{SndMsg} \;\mathsf{U}\; (\mathrm{RcvMsg} \wedge \mathrm{Rcv1})])])
\end{aligned}$$

taken from [40]. The formulae express the fact that sending a message (SndMsg) strictly alternates with receiving a message (RcvMsg), and that if a message with bit 0 (Snd0), resp. with bit 1 (Snd1), is sent, then a message with bit 0 (Rcv0), resp. with bit 1 (Rcv1), is received.

The Modal μ-Calculus The modal μ-calculus as introduced by Kozen [97] is one of the most powerful branching-time logics found in the literature. Based on mathematical fixpoint theory, it combines standard modal logic with least and greatest fixpoint operators. Similar to the λ-calculus for functional languages, and due to the extreme power of the fixpoint operators, on the one hand it allows us to express very complex (temporal) properties within a sparse syntactic formalism, while on the other hand, due to its syntactic minimalism, it also provides a good basis for the conceptual study of numerous theoretical intricacies which have attracted a lot of research. However, there is a price to be paid for these properties: even for experts, it is often hard to understand the meaning of a one-line μ-calculus formula. This is the reason for practitioners to prefer syntactically richer and less expressive derived logics. The μ-calculus

is nevertheless also practically relevant, as it provides a convenient "assembly language" for temporal logics.

The main characteristic which distinguishes the modal μ-calculus from most other branching-time logics is the possibility to specify *recursive* properties, which adds tremendous expressive power. Keeping in mind that the semantics of branching-time formulae are subsets of the state set $\mathcal{S}$, such recursive properties can be viewed as equation systems which have to be solved over $2^{\mathcal{S}}$. Although, in general, several solutions may exist there are two prominent solutions with respect to the subset ordering on state sets, namely the smallest and the largest ones. Due to a theorem of Tarski [140], their existence can be guaranteed whenever the equation system satisfies an easily verifiable monotonicity condition. The desired solution can then be expressed by means of a least or greatest fixpoint operator, where $\mu X.\phi(X)$, resp. $\nu X.\phi(X)$, denotes the smallest, resp. greatest, solution of $X = \phi(X)$.

More formally, the modal μ-calculus is an extension of Hennessy-Milner logic which introduces a (countable) set of variables *Var*, and a least fixpoint operator $\mu X.\phi$ binding X in ϕ. Its semantics is defined with respect to a valuation $\mathcal{V}$ mapping variables to subsets of $\mathcal{S}$. To ensure the above-mentioned monotonicity condition, it is customary to impose the syntactic restriction for expressions $\mu X.\phi$ that any occurrence of X in ϕ must occur within the scope of an even number of negations. The semantics for variables and the least fixpoint operator are then given as follows, where $\mathcal{V}[X \mapsto \mathcal{E}]$ is the valuation obtained from $\mathcal{V}$ by updating the binding of X to $\mathcal{E}$.

$$\begin{aligned} [\![X]\!]^{\mathcal{T}}_{\mathcal{V}} &=_{\mathrm{df}} \mathcal{V}(X) \\ [\![\mu X.\phi]\!]^{\mathcal{T}}_{\mathcal{V}} &=_{\mathrm{df}} \bigcap \{\, \mathcal{E} \subseteq \mathcal{S} \mid [\![\phi]\!]^{\mathcal{T}}_{\mathcal{V}[X \mapsto \mathcal{E}]} \subseteq \mathcal{E} \,\} \end{aligned}$$

The notions of bound and free occurrences of variables are defined in the usual way, and a formula is said to be closed if it does not contain any free variable occurrence. Using negation, it is possible to introduce also the greatest fixpoint operator $\nu X.\phi$ by

$$\nu X.\phi \;=_{\mathrm{df}}\; \neg\mu X.\neg\phi[\neg X/X]$$

where $\phi[\neg X/X]$ denotes the simultaneous replacement of all free occurrences of X by $\neg X$. The semantics of $\nu X.\phi$ can also be defined directly by

$$[\![\nu X.\phi]\!]^{\mathcal{T}}_{\mathcal{V}} \;=_{\mathrm{df}}\; \bigcup \{\, \mathcal{E} \subseteq \mathcal{S} \mid \mathcal{E} \subseteq [\![\phi]\!]^{\mathcal{T}}_{\mathcal{V}[X \mapsto \mathcal{E}]} \,\}$$

The clauses for the fixpoints are reformulations of the characterization in the Tarski-Knaster theorem [140] which states that the least fixpoint is the intersection of all pre-fixpoints and the greatest fixpoint is the union of all post-fixpoints. The fixpoint property also ensures that states satisfy a fixpoint formula iff they satisfy the *unfolding* of the formula, i.e.,

$$s \in [\![\sigma X.\phi]\!]^{\mathcal{T}}_{\mathcal{V}} \;\text{ iff }\; s \in [\![\phi[\,\sigma X.\phi/X\,]]\!]^{\mathcal{T}}_{\mathcal{V}} \;\text{ where } \sigma \in \{\,\mu,\nu\,\}$$

The following encoding of the until operators

$$\begin{aligned} \mathsf{E}[\phi \,\mathsf{U}\, \psi] &\equiv \mu X.\psi \vee (\phi \wedge (\textstyle\bigvee_{a \in \Sigma} \langle a \rangle X)) \\ \mathsf{A}[\phi \,\mathsf{U}\, \psi] &\equiv \mu X.\psi \vee (\phi \wedge (\textstyle\bigwedge_{a \in \Sigma} [a] X)) \end{aligned}$$

shows that the modal μ-calculus is strictly more expressive than CTL. Formulae resulting from such a translation are rather special as they possess only one sort of fixpoint, namely minimal fixpoints. The modal μ-calculus, however, permits users to write formulae containing alternating intertwined fixpoint operators, i.e., formulae like the following, which intuitively simply requires the existence of an infinite a-path on which the atomic proposition P is guaranteed to hold infinitely often.

$$\Phi_0 = \nu X.(\mu Y.(\langle a \rangle Y \vee (P \wedge \langle a \rangle X)))$$

Here it is characteristic that a fixpoint expression for a variable X contains another fixpoint expression of the other kind (e.g., greatest instead of least), which itself contains X free. Formulae with such alternations are not only hard to understand, but also hard to check algorithmically: already for finite-state systems, the best known algorithms are exponential in the so-called *alternation depth* which is defined as follows [123].

Definition 67 [Alternation Depth]
A formula Φ is said to be in the classes Σ_0 and Π_0 iff it contains no fixpoint operators. To form the class Σ_{n+1}, take $\Sigma_n \cup \Pi_n$, and close under (i) boolean and modal combinators, (ii) $\mu X.\Phi$, for $\Phi \in \Sigma_{n+1}$, and (iii) substitution of $\Phi' \in \Sigma_{n+1}$ for a free variable of $\Phi \in \Sigma_{n+1}$ provided that no free variable of Φ' is captured by Φ; and dually for Π_{n+1}. The (Niwinski) *alternation depth* of a formula Φ, denoted by $ad(\Phi)$, is then the least n such that $\Phi \in \Sigma_{n+1} \cap \Pi_{n+1}$, or equivalently $\Phi \in CBM(\Sigma_n \cup \Pi_n)$ where CBM denotes the closure under boolean and modal combinators.

Niwinski's inductive definition is stronger than the more popular and easier to understand definition of Emerson and Lei [49]. In order to illustrate Niwinski's quite technical definition, we give a few examples. The following subformulae of Φ_0 given above have, for instance, alternation depth 0, 1, and 2.

$$\begin{aligned} \langle a \rangle Y \vee (P \wedge \langle a \rangle X) &\in \Sigma_0 \cap \Pi_0 \\ \mu Y.(\langle a \rangle Y \vee (P \wedge \langle a \rangle X)) &\in \Sigma_1 \\ \nu X.(\mu Y.(\langle a \rangle Y \vee (P \wedge \langle a \rangle X))) &\in \Pi_2 \end{aligned}$$

The fragment of formulae with alternation depth one, called the *alternation-free modal μ-calculus*, has stirred up some greater interest. It captures a good deal of the desired system properties (in particular, the whole of CTL), while permitting efficient model checking: e.g., for finite-state systems model checking is linear in both the system and the formula size [45] (cf. Section 3.2). On the other hand, Bradfield [16] (and, independently, Lenzi[101]) showed that alternation depth induces an infinite sequence of sublogics of strictly increasing expressive power. This, and other interesting details about the modal μ-calculus and its associated theory can be found in chapter 1.4 on "Modal logics for Processes" in this handbook [17].

Presburger Computation Tree Logic (PCTL) It is well known that many relevant properties of reactive systems involve constraints on the numbers of occurrences of events. However, traditional temporal logics, like the ones we have considered so far, can only express regular properties of processes. This means that the set of models satisfying a given property form regular sets, and in particular in the branching-time setting are a regular set of trees. To overcome this limitation, Bouajjani, Echahed, and Robbana [10] introduced *Presburger Computation Tree Logic* (PCTL) which extends a propositional version of CTL with counting constraints on actions expressed by Presburger arithmetic formulae.

More formally, CTL is extended with state formulae π, i.e., boolean combinations of atomic propositions, Presburger arithmetic formulae f, existential quantification over integers written as $\exists x.\varphi$, and a binding operator $[x : \pi].\varphi$. The binding operator associates the variable x with the state formula π, and starting from the current state the variable x then counts the number of states satisfying π.

These new constructs permit to specify, for example, typical communication protocol properties like

> *"between the beginning and the end of every session, there are exactly the same numbers of requests and acknowledgements" [10]*

by the PCTL formula

$$\mathsf{AG}(\mathrm{BEGIN} \Rightarrow [x : \mathrm{REQ}].[y : \mathrm{ACK}].\mathsf{AG}(\mathrm{END} \Rightarrow (x = y)))$$

It has turned out, however, that PCTL is too expressive for automatic verification purposes, as it is undecidable already for finite-state systems. The authors therefore also introduce the fragment PCTL$^+$ which differs from PCTL in that the until operators must be of the restricted form $\mathsf{E}[\pi \,\mathsf{U}\, \phi]$ and $\mathsf{A}[\phi \,\mathsf{U}\, \pi]$ where π is required to be a state formula. Despite this syntactic restriction, PCTL$^+$ is still quite expressive and contains e.g., the formula given above. Moreover, it is decidable for BPA.

3.1.2 Linear-Time Logics

In contrast to branching-time logics, linear-time logics assume that at each moment in time there is only one determined possible future. This point of view concerning the semantics of time leads immediately to models where the semantic entities are sequences of events along a single time line, called *runs*. Depending on whether the underlying program structures may or may not contain deadlocks, i.e., states in which no further action is possible, these semantic models then either take into consideration both the finite, as well as the infinite runs, or restrict themselves to the infinite runs only.

In either case, formulae of a linear-time logic are then interpreted over the set of all (considered) runs, and the semantics $[\![\phi]\!]$ of a formula ϕ is the set of all runs π for which the formula holds, i.e., $[\![\phi]\!] = \{\, \pi \mid \pi \models \phi \,\}$.

Also in the case of linear-time logics, for system verification one is typically interested whether a specific state satisfies a certain property. This led to the

following convention: a state s of a transition system satisfies a formula if *every* run starting at s satisfies it.

In the following we introduce the linear-time logics most relevant for the study of infinite-state systems. For a better comparison with branching-time logics we will focus on *action-based* linear-time logics which have *relativised* next operators, one for each possible action.

Weak Linear Temporal Logic (WL) The weakest linear-time logic, called WL, is built in analogy to Hennessy-Milner logic out of *true*, negation, conjunction, and a relativised next operator $(a)\phi$, one for each action $a \in \Sigma$. Formally, WL formulae have the following syntax.

$$\phi ::= \mathtt{tt} \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid (a)\phi$$

The inductive definition below stipulates when a run π has the property ϕ, written as $\pi \models \phi$.

$$\begin{array}{ll} \pi \models \mathtt{tt} & \\ \pi \models \neg\phi & \text{if } \pi \models \phi \text{ does not hold} \\ \pi \models \phi_1 \wedge \phi_2 & \text{if } \pi \models \phi_1 \text{ and } \pi \models \phi_2 \\ \pi \models (a)\phi & \text{if } \pi(1) \xrightarrow{a} \pi(2) \wedge \pi^2 \models \phi \end{array}$$

Intuitively, 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.

Like for HML, we would like to remark that WL is mainly of theoretical interest due to its limited expressive power which only captures finite behaviour. In particular this means that the validity of a formula may readily be checked independently of the type of the considered (finite-branching) system by investigating all runs up to length $|\phi|$.

Linear Temporal Logic (LTL) *Linear Temporal Logic* (LTL) [127] can be seen as the "standard" linear-time logic. It is widely used in tools for automatic verification of reactive systems [105]. Similar to CTL, LTL is obtained by adding to WL an *until* operator $\phi \mathrel{U} \psi$. This extension increases the expressive power significantly, as it admits to reason about runs of unbounded length. The formal semantics of the until operator is given as follows.

$$\pi \models \phi \mathrel{U} \psi \quad \text{if } \exists\, i :\ \pi^i \models \psi \text{ and } \forall\, j < i :\ \pi^j \models \phi$$

Intuitively, a run $s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \ldots$ satisfies $\phi \mathrel{U} \psi$ if ϕ holds until eventually ψ holds, i.e., if there exists some suffix $s_i \xrightarrow{a_i} s_{i+1} \xrightarrow{a_{i+1}} \ldots$ satisfying ψ, and all preceding suffixes $s_j \xrightarrow{a_j} s_{j+1} \xrightarrow{a_{j+1}} \ldots$ with $j < i$ satisfy ϕ.

It is convenient to use the derived operators $F\phi = \mathtt{tt} \mathrel{U} \phi$ ("eventually ϕ") and its dual $G\phi = \neg F\neg\phi$ ("always ϕ"). They directly and concisely express simple liveness properties like "ϕ eventually happens" as $F\phi$, and simple safety properties like "ϕ never happens" as $G\neg\phi$.

Despite its expressive power, there are, however, comparatively basic properties like "at every *even* moment ϕ holds" that cannot be expressed in

LTL [146]. The reason for this inability becomes clear when switching to a formal language point of view. Obviously, the set of runs which satisfy a given LTL formula can also be interpreted as a formal ω-language, i.e., as a set of infinite words. Thomas [142] has shown that the LTL definable ω-languages correspond to the class of star-free ω-regular languages, an interesting subclass of the ω-regular languages which results from restricting the monadic second order logic over infinite words (S1S) to first-order logic.

Because of this lack of expressiveness of LTL, various more expressive logics have been studied, most notably the linear-time μ-calculus which we consider next.

The Linear-Time μ-Calculus (LTμ) The *linear-time μ-calculus* [144] is the linear-time analogue to the modal μ-calculus: it is a powerful fixpoint logic in which all the usual linear-time operators like "always", "eventually", and "until" can be described. In particular, the logic is equivalent in its expressive power to Büchi automata [144, 47], and hence also to monadic second order logic over infinite words (S1S) [18]. Consequently, the linear-time μ-calculus characterizes the full class of ω-regular languages and is thus strictly more expressive than LTL which may only describe star-free ω-regular languages.

This expressiveness is gained from adding variables to WL, together with a least fixpoint operator $\mu X.\phi$, which binds the variable X. In order to ensure vital monotonicity properties, X is only allowed to occur in ϕ within the scope of an even number of negations.

The formal development proceeds structurally similarly as in the branching-time case. The semantics of the linear-time μ-calculus is defined with respect to a valuation $\mathcal{V}$ mapping variables to sets of runs. The denotations for variables and the fixpoint operator are inductively defined by the following rules, where $\mathcal{R}$ denotes the set of all runs of a given labelled transition system.

$$\begin{array}{lcl} [\![X]\!]_{\mathcal{V}} & = & \mathcal{V}(X) \\ [\![\mu X.\phi]\!]_{\mathcal{V}} & = & \bigcap \{\, R \subseteq \mathcal{R} \mid R \subseteq [\![\phi]\!]_{\mathcal{V}[R \mapsto X]} \,\} \end{array}$$

Monotonicity arguments guarantee that $[\![\mu X.\phi]\!]_{\mathcal{V}}$ is the least fixpoint of the function which assigns to a set R of runs the set $[\![\phi]\!]_{\mathcal{V}[R \mapsto X]}$. Dual operators can be defined just as in the branching-time case, allowing to transform every formula into positive normal form, which is very convenient for verification purposes.

Finally, the alternation depth of LTμ formulae is defined as for the modal μ-calculus. From a semantic point of view, there is, however, an important difference, as alternation depth does not yield a hierarchy of strictly more expressive sublogics. This follows from the fact that LTμ is equally expressive as Büchi automata, and a Büchi automaton can be encoded by an LTμ formulae of alternation depth at most two [124].

Constrained Linear Temporal Logic (CLTL) The most expressive linear-time logic considered here is *Constrained Linear Temporal Logic* (CLTL), which

was introduced in [13]. In analogy to PCTL (see Section 3.1.1) it allows to define nonregular properties, i.e., properties which characterize nonregular sets of runs, and that are therefore not definable by finite-state ω-automata. This expressiveness is achieved by extending LTL with two kinds of constraints: *counting constraints*, which use Presburger arithmetic formulae to express constraints on the number of occurrences of events, and *pattern constraints*, which use finite-state Rabin-Scott automata to impose structural constraints on runs.

More formally, LTL is first extended with state formulae π, i.e., boolean combinations of atomic propositions.

Counting constraints are then introduced, as in the case of PCTL, by Presburger arithmetic formulae f, existential quantification over integers written as $\exists x.\varphi$, and a binding operator $[x : \pi].\varphi$. Through the binding operator the variable x is associated with the state formula π, and from the current state on the variable x then counts the number of states satisfying π. For example, the formula

$$[x_1 : \pi_1].[x_2 : \pi_2].G(P \Rightarrow (x_1 \leq x_2))$$

specifies the property that in every computation starting from the current state we have the invariant "whenever P holds, the number of states satisfying π_2 is greater or equal than the number of states satisfying π_1".

Finally, pattern constraints are introduced using formulae of the form $\downarrow u.\varphi$, and A^u where A is a deterministic finite-state automaton. The intended meaning is that a formula $\downarrow u.\varphi$ associates the current state with the position variable u, and if A^u is a subformula of φ, then A^u holds whenever the trace of the computation from the state u to the current state is accepted by the automaton A. For instance, the formula

$$\downarrow u.[x_1 : \pi_1].[x_2 : \pi_2].G(A^u \Rightarrow (x_1 \leq x_2))$$

means that in every computation starting from the current state which is accepted by A, the number of states satisfying π_2 is greater than or equal to the number of states satisfying π_1.

As in the branching-time case, already for finite-state systems this combination of counting and pattern constraints yields undecidability. On the other hand, two important fragments of CLTL which still allow us to specify a wide range of nonregular properties while retaining decidability have been identified: $\text{CLTL}_\Box$, where counting constraints cannot be introduced in eventuality formulae, and the less expressive simple-$\text{PLTL}_\Box$, which extends LTL with counting constraints, where only propositional formulae may be used in eventuality formulae.

$\text{CLTL}_\Box$, and therefore also CLTL, are strictly more expressive than LTL, since, e.g., the typical property of runs "every finite prefix contains at least as many a actions as b actions" is expressible in $\text{CLTL}_\Box$ but not even in the linear-time μ-calculus. On the other hand, simple-$\text{PLTL}_\Box$ is incomparable to LTL, but allows us to express the complement of simple ω-regular languages, which are ω-languages definable by a finite-state Büchi automaton where every loop in its transition graph is a self-loop.

3.2 Algorithms for Branching-Time Logics

In the branching-time setting we have to distinguish two rather different situations.

First, for sequential type infinite-state systems like BPA, PDA and some extensions thereof monadic second order logic (MSOL) is known to be decidable by a complicated reduction to the emptiness problem for regular languages on trees definable in monadic second order logic with n successors (SnS) [122]. As a consequence, the modal μ-calculus which can be strictly embedded in MSOL is also decidable, and may thus be used effectively for the specification of properties for these systems. A drawback is, however, that all decision procedures based on this approach are nonelementary. To palliate this problem in the last years some simpler and more direct algorithms have been developed which will be presented in Section 3.2.1 and 3.2.2.

Second, for infinite-state models of concurrent computation like BPP, MSA, or Petri nets already the weak logic EG is undecidable [56]. This is mainly due to the possibility of these models to describe grid-like structures, an easy cause of undecidability. The only positive result for this class of infinite-state models concerns the logic EF and the process class BPP [52] which turn out to impose a PSPACE-complete model checking problem [109, 111].

3.2.1 BPA

Hennessy-Milner Logic Since BPA processes are finite-branching, model checking HML is trivially decidable. As for all finite-branching processes, the given BPA process has only to be unfolded up to depth $|\phi|$, for a HML formula ϕ at hand, as validity of ϕ can then readily be checked. The encoding of the unfolded tree as a BPA process is, however, rather concise. More concretely, Mayr [112] has shown that the general model checking problem for HML and BPA is PSPACE-complete by a reduction from the problem of quantified boolean formulae (QBF).

	General	Fixed Formula
HML	PSPACE-complete	PTIME

The Modal μ-Calculus The combination of expressive power and syntactic minimality makes the modal μ-calculus an ideal formalism for conceptual studies. It is therefore not surprising that it is indeed the most studied logic for BPA processes and its extensions. In particular, this line of research has produced several different model checking algorithms which can be classified into being either *global* or *local* in nature, and into algorithms dealing with the full logic or only with a fragment thereof.

The first algorithm given for BPA processes was a global one handling the alternation-free fragment [27]. Although the original algorithm works on formulations of BPA processes and alternation-free formulae which are rather different from those used in this survey, an adaptation to our setting is straightforward. The model checking algorithm proceeds by iteratively computing a formula specific *property transformer* for each nonterminal of the given BPA system. To

be more precise, let ϕ be a fixed μ-calculus formula and D_ϕ be the power set of subformulae of ϕ. A property transformer T_A for a nonterminal A is then a mapping $D_\phi \longrightarrow D_\phi$ where $T_A(\Delta) = \Delta'$ means that Δ' is the set of subformulae of ϕ valid at A under the assumption that all subformulae in Δ are valid after termination of A, i.e., at the final state ε from which no action can occur. Once the property transformers have been computed by means of a fixpoint iteration, 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 subformulae of ϕ satisfied by ε (which can be easily *locally* computed as ε has no successor states).

Let us illustrate this approach further by a simple example.

Example 8 *We take as the system to be model checked the BPA system from Example 1 in Section 1.1 which is defined by the following three rewrite rules*

$$X \xrightarrow{a} XB,\ X \xrightarrow{c} \varepsilon,\ B \xrightarrow{b} \varepsilon$$

For this system we want to automatically verify that from the root X there exists an infinite a-path on which the action c is always enabled. This property can formally be expressed by the alternation-free modal μ-calculus formula

$$\phi =_{df} \nu Z.\langle a\rangle Z \wedge \langle c\rangle tt$$

The first step of the second-order model checking algorithm consists now of transforming the formula into an equivalent equational form [43] which explicitly names all subformulae, yielding in our case

$$\nu\{ Z_1 = Z_2 \wedge Z_3,\ Z_2 = \langle a\rangle Z_1,\ Z_3 = \langle c\rangle Z_4,\ Z_4 = tt\}$$

Defining $\mathcal{Z} =_{df} \{ Z_1, Z_2, Z_3, Z_4 \}$, the property transformers to be computed for the nonterminals X and B are thus mappings $D_\phi \longrightarrow D_\phi$ where $D_\phi = 2^{\mathcal{Z}}$. For notational convenience, we split these property transformers further into component property transformers $T_N^{Z_i} : D_\phi \longrightarrow 2^{\{Z_i\}}$, for $i \in \{1,\ldots,4\}$ and $N \in \{X, B\}$. These component transformers are just the projections on the Z_i, i.e., $T_N^{Z_i}(M) =_{df} t^{Z_i}(T_N(M))$ where $t^{Z_i}(M) =_{df} \{Z_i\}$ if $Z_i \in M$, and $t^{Z_i}(M) =_{df} \emptyset$ otherwise. Hence we have $T_N(M) = \bigcup_{i=1}^4 T_N^{Z_i}(M)$. The next step combines now the BPA system with the formula in equational form yielding the following equations for the component property transformers

$$\nu \left\{ \begin{array}{ll} T_X^{Z_1} = T_X^{Z_2} \sqcap T_X^{Z_3} & T_B^{Z_1} = T_B^{Z_2} \sqcap T_B^{Z_3} \\ T_X^{Z_2} = T_X^{Z_1} \circ T_B & T_B^{Z_2} = t^{\perp} \\ T_X^{Z_3} = t^{Z_4} & T_B^{Z_3} = t^{\perp} \\ T_X^{Z_4} = t^{\top} & T_B^{Z_4} = t^{\perp} \end{array} \right\}$$

where $\sqcap$ means argumentwise intersection, and $\circ$ functional composition. Furthermore, we use $t^{\top}$, respectively $t^{\perp}$, to denote the function which maps every subset of D_ϕ to $\mathcal{Z}$, respectively $\emptyset$.

As we are looking for the largest solution, we have to initialize every component transformer with $\mathsf{t}^\top$*, and get the following fixpoint iteration.*

	0	1	2	3
$\mathsf{T}_X^{Z_1} = \mathsf{T}_X^{Z_2} \sqcap \mathsf{T}_X^{Z_3}$	$\mathsf{t}^\top$	$\mathsf{t}^\top$	t^{Z_4}	t^{Z_4}
$\mathsf{T}_X^{Z_2} = \mathsf{T}_X^{Z_1} \circ \mathsf{T}_B$	$\mathsf{t}^\top$	$\mathsf{t}^\top$	$\mathsf{t}^\top$	$\mathsf{t}^\top$
$\mathsf{T}_X^{Z_3} = \mathsf{t}^{Z_4}$	$\mathsf{t}^\top$	t^{Z_4}	t^{Z_4}	t^{Z_4}
$\mathsf{T}_X^{Z_4} = \mathsf{t}^\top$	$\mathsf{t}^\top$	$\mathsf{t}^\top$	$\mathsf{t}^\top$	$\mathsf{t}^\top$
$\mathsf{T}_B^{Z_1} = \mathsf{T}_B^{Z_2} \sqcap \mathsf{T}_B^{Z_3}$	$\mathsf{t}^\top$	$\mathsf{t}^\top$	$\mathsf{t}^\bot$	$\mathsf{t}^\bot$
$\mathsf{T}_B^{Z_2} = \mathsf{t}^\bot$	$\mathsf{t}^\top$	$\mathsf{t}^\bot$	$\mathsf{t}^\bot$	$\mathsf{t}^\bot$
$\mathsf{T}_B^{Z_3} = \mathsf{t}^\bot$	$\mathsf{t}^\top$	$\mathsf{t}^\bot$	$\mathsf{t}^\bot$	$\mathsf{t}^\bot$
$\mathsf{T}_B^{Z_4} = \mathsf{t}^\bot$	$\mathsf{t}^\top$	$\mathsf{t}^\top$	$\mathsf{t}^\top$	$\mathsf{t}^\top$

The final step consists of two parts. First, computing the set of all subformulae of Φ *valid for* ε*, which in this example gives* $\{ Z_4 \}$*. Second, applying the property transformer of* X *to* $\{ Z_4 \}$*, which results in*

$$\mathsf{T}_X(\{ Z_4 \}) = \mathcal{Z}$$

tells us that, in particular, Z_1 *and therefore* Φ *holds at* X*. Thus, as one might have expected, the considered system does indeed have an infinite path from* X *on which the action* c *is always enabled. For more details about second-order model checking see [21].*

This second order algorithm has been generalized in [29] to handle the full modal μ-calculus. Although the actual model checker results directly from a combination of the alternation-free variant and some kind of backtracking, as known from finite-state model checking algorithms (cf. [44]), the corresponding correctness proof requires a stronger framework, which uses *dynamic environments* [29]. They are needed to explicitly model valuations of *free* variables during the iteration process.

Both model checking algorithms have in common that they are *global*, as they provide complete information about which of the subformulae of Φ are satisfied by the states of the BPA process under consideration. It is, in fact, even possible to explicitly represent the set of all states S_Φ of a given transition system satisfying a given formula Φ by means of a finite automaton constructed from the obtained property transformers. In particular, this shows that S_Φ is always a regular set over the set of nonterminals [21].

The two algorithms have a time complexity which is *polynomial* in the size of the system, while *exponential* only in the size of the formula. From a practical point of view this means that the verification of BPA processes should be still feasible, since realistic system properties can usually be expressed by small formulae.

Mayr [112] has shown that this complexity is essentially optimal by giving a matching lower bound for it. He obtained this lower bound by reducing the

acceptance problem for linearly space bounded alternating Turing-machines to model checking BPA and the alternation-free μ-calculus.

Summarizing, it follows that model checking the modal μ-calculus, or its alternation-free fragment, for BPA is EXPTIME-complete.

	General	Fixed Formula
alternation-free modal μ-calculus	EXPTIME-complete	PTIME
full modal μ-calculus	EXPTIME-complete	PTIME

Hungar and Steffen [78] present a local model checking alternative to [27] in 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 $A \vdash \langle \phi, \Delta \rangle$, with the intended meaning that ϕ holds at state A under the assumption that all formulae of Δ hold at the final state. The heart of the tableau system is the following composition rule, which is 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 show that ϕ holds at $\alpha\beta$ under the assumption that Δ holds at the final state, we guess an *intermediate assertion* Γ, and prove the following:

- if Δ holds at the final state, then Γ holds before the execution of β;
- if Γ holds after execution of α, then ϕ holds immediately before its execution.

Hungar [75] extended this tableau system for parallel compositions of BPA processes and the alternation-free modal μ-calculus. In contrast to BPP where no synchronization is possible between processes, he considers a model where BPA processes communicate on common actions. The tableau system presented is sound, but, in general, not complete, 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 [28] 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.

The Logic PCTL Finally, we mention that the verification of nonregular properties for BPA processes has also been considered. Bouajjani, Echahed and Robbana introduce in [10] the logic PCTL, an extension of CTL with Presburger arithmetic (see Section 3.1.1). First of all they show that the logic PCTL is undecidable even for finite-state systems. This negative result is proved by a reduction to the halting problem for Minsky machines and underpins the expressive power of PCTL. Weakening, subsequently, PCTL to its fragment PCTL$^+$ they regain on the other hand decidability of the model checking problem by reduction to the validity problem of Presburger arithmetic.

3.2.2 Pushdown Processes and Extensions

Verifying Hennessy-Milner Logic is essentially independent of the considered process class, as long as the transitions systems which have to be considered are finite-branching. Moreover, the considered logics expressing nonregular properties are too powerful to be decidable for the class of pushdown processes and even more expressive models. Our discussion of the process classes beyond BPA will therefore focus on the alternation-free and full μ-calculus.

Pushdown Processess In formal language theory, automata are used as a "device" to accept sets of words. This goal leaves the free choice between two equivalent acceptance criteria: acceptance with final control states and acceptance with empty stack. For process theory, however, where automata are considered as a general computational model, acceptance with empty stack provides a much better conceptual match: Reaching a PDA configuration where the stack is empty then simply corresponds to termination of the process associated with the considered pushdown automaton.

After the observation that the process class PDA, the class of transition graphs defined by pushdown automata accepting with empty stack, is strictly more expressive than the class BPA [34] there has been a spurt of activity in the development of verification algorithms for such pushdown automata and their extensions.

Roughly, these algorithms can be classified as being

- global, iterative, or
- based on games, or
- based on reachability analysis.

The first algorithm developed for PDA [28] can handle the alternation-free μ-calculus and is a generalization of the global, iterative one for BPA [27]. It takes into account the finite control $Q = \{ q_1, \ldots, q_n \}$ of the given pushdown automaton, by observing that starting from a configuration qA the PDA can terminate in any of the configurations $q_1\epsilon, \ldots, q_n\epsilon$ representing the empty stack. As a consequence, the property transformer for the nonterminal A must now be parameterized by $q \in Q$, and instead of a single assertion the property transformer must take now $|Q|$ assertions into account, one for each of the terminating configurations $q_i\epsilon$. Besides this more general domain for the property transformers the algorithm is similar to [27], and can be extended along the lines of [29] to handle also the full modal μ-calculus.

A second, quite different approach is to consider games on the transition graphs of PDAs. Walukiewicz has shown that model checking a modal μ-calculus formula ϕ for a PDA P can be reduced to a parity game on a pushdown tree obtained from the combination of P and the syntax tree for ϕ [149]. Furthermore, he proved that the winning strategy for a player in such a game can be realized by a pushdown strategy automaton. Using this strategy automaton, it is then possible to reduce the existence of a winning strategy in the pushdown game to the existence of a winning strategy in a finite parity game associated

with the strategy automaton. The last problem can, finally, be decided by any model checking algorithm for finite-state systems. It has to be pointed out, however, that the size of the finite game constructed is exponential in the size of the pushdown automaton. More specifically, Walukiewicz has shown that model checking the alternation-free μ-calculus for PDAs is EXPTIME-complete, and that this result even holds for a fixed formula.

The third technique for model checking pushdown automata is to analyze reachable sets of configurations as introduced by Bouajjani, Esparza, and Maler [11]. Their approach is based on *alternating pushdown automata*, a generalization of the classical notion of PDA which admits AND/OR transition steps. The main idea of the algorithm is, given a PDA P and an alternation-free μ-calculus formula ϕ, to construct the product of P and ϕ by means of an alternating pushdown automaton $\mathsf{AP}_{\mathsf{P}\times\phi}$. This representation allows us to exploit that for alternating pushdown automata the set of all predecessors $\mathsf{pre}^*(\mathsf{C})$ for a regular set of configurations C is always regular and can effectively be computed in terms of alternating finite-state automata. In the case of $\mathsf{AP}_{\mathsf{P}\times\phi}$ this means that starting with the regular set $\mathsf{C}_{\mathtt{tt}}$ of all pairs of configurations c and atomic formulae ψ such that $\mathsf{c} \models \psi$, it is possible to effectively compute the set of all configurations which satisfy ϕ, and hence to solve the model checking problem.

The known complexity results for model checking PDA are summarized in the following table.

	General	Fixed Formula
alternation-free modal μ-calculus	EXPTIME-complete	EXPTIME-complete
full modal μ-calculus	EXPTIME-complete	EXPTIME-complete

Regular Graphs Although we have emphasized in this survey the rewriting-based approach for the description of infinite transition systems, other natural representations have also been considered. Especially in graph theory (cf. [8]), a variety of graph generation mechanisms have been studied. A particularly attractive candidate in this field is the framework of deterministic graph grammars. They consist of an initial finite hypergraph together with a set of hyperedge replacement rules where hyperedges play the roles of nonterminals, and finite hypergraphs the roles of right-hand side expressions. A particular deterministic graph grammar represents then a single infinite graph which is obtained as the limit of the finite expansions of the initial hypergraph.

Figure 5 shows an example graph grammar with a single rule where the initial hypergraph consists of the single edge X.

Graph grammars characterize the class of *regular graphs* which is equally expressive as the class of pushdown automata when unguarded rewrite rules are allowed [32]. Nevertheless, Courcelle has shown that regular graphs still possess a decidable monadic second order theory [46]. As already for BPA, this decidability result yields, however, only a nonelementary complexity upper bound.

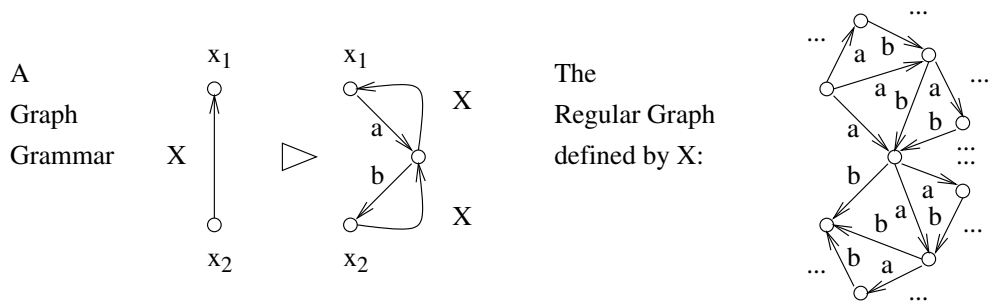


Figure 5: An example graph grammar.

Subsequently, Burkart and Quemener [26] have developed a direct model checking algorithm for the alternation-free μ-calculus and regular graphs which generalizes the global, iterative model checking algorithm of [27, 28]. They extend the method of computing property transformers for nonterminals to whole finite hypergraphs where the glue vertices play the roles of start and end states. Surprisingly, this generalization has the same worst-case complexity as model checking pushdown automata. Along the lines of [29] the algorithm can even be adapted to cope with the full modal μ-calculus.

$\mathbf{REC_{Rat}}$ Graphs A still more expressive class of transition systems was introduced by Caucal in [33]. 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. A rewrite rule $L_1 \xrightarrow{a} L_2$ is then interpreted as the infinite collection of "ordinary" rules $\alpha \xrightarrow{a} \beta$ where α is a word of the regular language L_1 and β is a word of the regular language L_2. Since these rules may only be applied with respect to prefix rewriting, $\mathrm{REC_{Rat}}$ belongs to an extension of sequential rewrite transition systems in which the set of rules may be infinite. Caucal's 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.

By exploiting an alternative representation of $\mathrm{REC_{Rat}}$ graphs in terms of inverse regular morphisms and regular restrictions also given in [33] Burkart [22] has developed a more direct model checking algorithm for the full modal μ-calculus. The idea is to combine the given formula ϕ, the inverse regular morphism h^{-1}, and the regular restriction L into a new μ-formula $\Phi(\phi, h^{-1}, L)$ which has then to be checked for validity on an infinite, complete tree with backward edges. The latter problem can readily be encoded as a BPA model checking problem for the full modal μ-calculus, thereby resulting in a single exponential model checking algorithm for $\mathrm{REC_{Rat}}$.

Macro Processes Another natural extension of BPA are *macro processes* as considered by Hungar in [76]. This framework extends the interpretation of BPA processes as procedures, first given in [27], by allowing transitions with

"higher-order procedure calls". Macro processes add to the simple idea that nonterminals correspond to procedure identifiers and right-hand sides to procedure bodies the concept of typed formal procedure parameters: left-hand sides in the grammar have typed formal parameters, which can be instantiated by nonterminals of appropriate type.

The typing discipline establishes a strict "is allowed to be passed as a parameter to" hierarchy among the nonterminals, which, in particular, excludes the possibility of self application: it is not possible to pass a procedure to itself as a parameter. This is essential to maintain the decidability of the model checking problem.

The concept of higher-order recursion is quite powerful. In fact, the class of macro processes belonging to the first level in the hierarchy corresponds to the class of potentially infinite-branching pushdown processes, and thus covers the class of regular graphs, while macro processes of higher levels are strictly more expressive [77].

For these macro processes, Hungar developed a local as well as global iterative model checking algorithm for the alternation-free μ-calculus. Both algorithms have complexity $O(\mathsf{tower}(k))$ for a type hierarchy of depth k where $\mathsf{tower}(0) =_{df} 0$ and $\mathsf{tower}(n+1) =_{df} 2^{\mathsf{tower}(n)}$. In particular, this shows that his algorithm for the general problem, where k remains unspecified, has nonelementary time complexity.

3.2.3 BPP and Petri Nets

Going from models for sequential computation to models for concurrent computation the picture concerning decidability of model checking changes dramatically in the branching-time setting.

The Logic EG In [56], Kiehn and Esparza show that the model checking problem for the logic EG is undecidable for BPP, and consequently, also for all the other considered branching-time logics except EF. This result is obtained by a reduction from the halting problem for Minsky two counter machines which is known to be undecidable [117]. Given a Minsky two counter machine M, it is possible to construct a BPP P_M which models M in the weak sense that P_M may simulate the computations of M, but may also "cheat" at some points during the simulation. Nevertheless, it is still possible to characterize the "honest runs" of the system, as well as reachability of the halting state, in the logic EG, although the formula becomes quite complicated. A closer inspection of the proof reveals that the result even holds for the restricted class of deterministic BPPs, and that the formula expressing that the halting state can be reached along an honest computation is independent of the argument process.

The Logic EF The only branching-time logic not covered by the negative result for the logic EG is the logic EF. For this logic Esparza has shown in [52] that the model checking problem for BPPs is decidable. The result is based on the fact that the reachability relation for BPPs is effectively semilinear, i.e.,

can be characterized by a formula in Presburger arithmetic. Combining then this Presburger formula with the given EF formula ϕ yields again a Presburger formula characterizing the set of reachable states satisfying ϕ. Based on a decision procedure for the validity of Presburger arithmetic formulae one obtains a double exponential time model checking algorithm for EF and BPPs.

On the other hand, Esparza also proved a lower bound for the problem by a reduction of the validity problem for quantified boolean formula (QBF), which is known to be PSPACE-complete. This PSPACE-hardness result, which even holds for BPPs that describe finite-state systems, was complemented by Mayr [109, 111] who showed that the model checking problem for full BPP only requires polynomial space, thereby establishing that it is PSPACE-complete. Thus adding recursion to finite-state BPPs does not increase the complexity of model checking. A further consequence of Mayr's result is that for a fixed formula of nesting depth k the problem lies in Σ_k^P, the k-th level of the polynomial hierarchy. Overall, model checking BPP's with respect to EF has the following complexities.

	General	Fixed Formula
EF	PSPACE-complete	Σ_k^P

Going up in the process hierarchy to the more expressive class of Petri Nets costs decidability for EF as shown by Esparza in [51]. Undecidability is shown by a reduction from the undecidable containment problem for Petri Nets, which is defined as follows: determine for every two Petri Nets N_1 and N_2 with the same number of places and each bijection f between their places, whether for each reachable marking M of N_1 it is guaranteed that $f(M)$ is reachable in N_2.

3.2.4 PA

As PA properly contains BPP, the logic EG is immediately undecidable for PA. Hence, the only branching-time logic not covered by this result is the logic EF for which Mayr [110] has proved decidability. He presents a sound and complete tableau system which is based on process decomposition. This algorithm has complexity $O(tower(k))$ for formulae of nesting-depth k. Moreover, when the formula is fixed, the complexity lies in Σ_k^P. Later, Lugiez and Schnoebelen [104] proved decidability of the logic EF for PA by a completely different method using tree-automata to represent sets of configurations. The complexity of their algorithm is, however, the same as that of Mayr's algorithm. Since the best known lower bound for the problem is PSPACE-hardness, the exact complexity of this model-checking problem is still unknown.

	General	Fixed Formula
EF	PSPACE-hard	Σ_k^P

3.2.5 General Processes

We close this section by mentioning that Bradfield has proposed in [15] a sound and complete tableau system for the modal μ-calculus and arbitrary infinite

transition systems. This tableau system is of course highly undecidable, but provides a general guideline for interactive proofs.

3.3 Algorithms for Linear-Time Logics

From a theoretical point of view linear-time logics are quite attractive as they have a natural connection to automata on infinite strings. More specifically, the set of runs denoted by formulae of classical linear-time logics like e.g., LTL or the linear-time μ-calculus are always ω-regular sets. Consequently, most of the model checking algorithms for linear-time logics follow the automata-based approach where linear-time formulae are modelled as Büchi automata [146, 145]. Intuitively this approach works as follows. Given a process and a linear-time formula ϕ a Büchi automaton $\mathcal{B}_{\neg\phi}$ is constructed, which accepts exactly all runs not satisfying ϕ. Building the product of this Büchi automaton with the Büchi automaton for the process yields a Büchi automaton, whose language is empty, if and only if it satisfies Φ. This reduces the model checking problem to the emptyness problem for Büchi automata.

Two different methods for the construction of a "property automaton" $\mathcal{B}_\phi$ have been proposed.

The first method could be termed *global*, as it constructs two automata $\mathcal{B}_\mathsf{L}$ and $\overline{\mathcal{B}_\mathsf{R}}$ such that $\mathcal{L}(\mathcal{B}_\phi) = \mathcal{L}(\mathcal{B}_\mathsf{L}) \cap \mathcal{L}(\overline{\mathcal{B}_\mathsf{R}})$. Here the automaton $\mathcal{B}_\mathsf{L}$ checks some local conditions on the model, while $\overline{\mathcal{B}_\mathsf{R}}$ is the complement automaton of $\mathcal{B}_\mathsf{R}$ which checks for non-well-foundedness of a regeneration sequence related to minimal fixpoints [144].

The second method takes the *compositional* approach where with each connective in the logic a corresponding construction on automata is associated. The final automaton $\mathcal{B}_\phi$ is then obtained by starting with the basic automata for the variables in ϕ, and applying the corresponding automata constructions inductively according to the structure of ϕ. The main difficulty of this method, which has been developed by Dam for the linear-time μ-calculus [47], is concerned with the automata constructions related to the fixpoint operators.

All algorithms considered in the sequel follow the automata-based approach and may use either the global or the compositional construction for the property Büchi automaton. Since Büchi automata are equally expressive as the linear-time μ-calculus all algorithms directly deal with the more expressive linear-time μ-calculus, and subsume model checking LTL as a special case.

Before we proceed with the specifics of the infinite-state case, we mention that LTL, respectively LTμ, model checking for finite-state systems has intensively been studied. For instance, it is well-known that model checking LTL, respectively LTμ, is PSPACE-complete for finite-state systems [135, 144]. Both results follow from a polynomial reduction to satisfiability testing which is shown to be PSPACE-complete itself. Fortunately, the complexity is exponential only in the size of the formula which is in practice often small, but linear in the system size [102]. Despite these at first sight negative results, the feasibility of linear-time model checking for finite-state systems has been demonstrated by practical verification tools like SPIN [73], PROD [147] or PEP [148].

3.3.1 BPA and Pushdown Processes

LTL and LTμ The decidability of the model checking problem for pushdown processes and the linear-time μ-calculus follows immediately from the decidability of the corresponding problem for the modal μ-calculus, since the linear-time μ-calculus can be interpreted in its branching-time equivalent, however at the price of an exponential blowup. This blowup in the translation can, however, be avoided by a more direct approach which applies automata-theoretic techniques.

Given a PDA $\mathcal{P}$ and a linear-time formula ϕ, one constructs as usual a Büchi automaton $\mathcal{B}_{\neg\phi}$ accepting exactly all runs which do not satisfy ϕ. Combining then this Büchi automaton with the PDA $\mathcal{P}$ yields a Büchi PDA $\mathcal{P} \times \mathcal{B}_{\neg\phi}$ representing the synchronized product of $\mathcal{P}$ and $\mathcal{B}_{\neg\phi}$ where a run is accepted if it visits a certain set of control locations infinitely often. Accordingly, all runs starting in a configuration c of $\mathcal{P}$ satisfy ϕ iff c has no accepting run in $\mathcal{P} \times \mathcal{B}_{\neg\phi}$. Bouajjani, Esparza, and Maler have shown in [11] that the set $\mathsf{C}_{\neg\phi}$ of all configurations of this Büchi PDA possessing an accepting run can effectively be computed by means of a certain kind of finite automaton, called *multi-automaton*. The model checking problem then simply reduces to checking whether the initial configuration of the PDA $\mathcal{P}$ is a member of $\mathsf{C}_{\neg\phi}$.

This algorithm shows that model checking pushdown processes with LTL, as well as the linear-time μ-calculus, is in EXPTIME. A reduction of the acceptance problem for linearly bounded alternating Turing machines, which is known to be EXPTIME-complete [93], shows, furthermore, that the problem is EXPTIME-hard. Overall, these results establish that the model checking problem for pushdown automata wrt. LTL, as well as the linear-time μ-calculus, is EXPTIME-complete. Fixing a formula ϕ, on the other hand, the model checking problem has only polynomial complexity in the size of the system representation. Thus the problem is only slightly harder than in the finite-state case which is PSPACE-complete, and it is also polynomial in the system size for a fixed formula.

Shortly afterwards Mayr [111] has also settled the exact complexity of model checking LTL for BPA processes which is not covered by the previous results. He has shown, by generalizing the proof of Bouajjani, Esparza, and Maler, that model checking the weaker class of context-free processes with LTL is EXPTIME-complete as well.

Summarizing, we have the following complexity results for model checking LTL, respectively LTμ, on sequential processes.

	General	Fixed Formula
FSA	PSPACE-complete	PTIME
BPA	EXPTIME-complete	PTIME
PDA	EXPTIME-complete	PTIME

In addition, Esparza et al. [54] have developed efficient algorithms for model checking pushdown processes when the negation of the specification is already

given in form of a Büchi automaton. This circumvents the exponential conversion from LTμ to Büchi automata.

The Logic CLTL Finally, decidability of nonregular linear-time properties has been investigated by Bouajjani and Habermehl in [12]. They consider CLTL, a linear-time logic which extends LTL with pattern constraints expressed by finite-state automata and counting constraints using Presburger arithmetic formulae. As full CLTL is undecidable even for finite-state systems [13], its fragment $\text{CLTL}_\Box$ which essentially does not allow eventuality formulae with counting constraints is introduced. It turns out that $\text{CLTL}_\Box$ is decidable for pushdown automata which is shown in four steps:

- First, since $\text{CLTL}_\Box$ is not closed under negation, the syntactic complement of $\text{CLTL}_\Box$, called $\text{CLTL}_\Diamond$, is introduced. Both fragments are related by the property that for each $\phi \in \text{CLTL}_\Box$ there exists $\phi' \in \text{CLTL}_\Diamond$ such that $\neg\phi$ and ϕ' have the same semantics, and vice versa.

- Second, it is shown that any formula of $\text{CLTL}_\Diamond$ can be transformed into a normal form which is essentially a conjunction of an ω-regular property expressed in terms of a linear-time μ-calculus formula and a pure counting constraint eventuality formula.

- Third, it is proved that satisfiability of a formula $\phi \in \text{CLTL}_\Diamond$ relatively to a set of sequences S can be reduced to a constrained emptiness problem. If $\tilde{S}$, a technical set construction related to S, has the additional property that its intersection with an ω-regular language always yields a semilinear set, then the satisfiability problem can further be reduced to the nonemptiness problem of semilinear sets which itself is known to be decidable.

- Fourth and last, the fact that pushdown processes generate ω-context-free languages, which are semilinear and closed under intersection with ω-regular languages shows that the satisfiability problem for $\text{CLTL}_\Diamond$ is decidable, and consequently so is the model-checking problem for $\text{CLTL}_\Box$.

3.3.2 BPP and Petri Nets

LTL and LTμ Here, only the model checking problem for the more expressive class of Petri nets and the stronger logic LTμ has been considered. As it turns out, the decidability of the model checking problem for LTμ and Petri nets is sensitive to whether atomic propositions beyond just *true* are allowed in the logic or not. As shown by Esparza, the case with general atomic propositions is undecidable [53], while in case that *true* is the only allowed atomic proposition this problem becomes decidable [51].

The given decision procedure is based on an automata-theoretic characterisation of the logic which was introduced by Vardi and Wolper [146]. Refining a technique of [47] Esparza has shown that, given a closed formula φ, there

exist two finite automata $A_{\neg\phi}$ and $B_{\neg\phi}$ accepting the finite, respectively infinite, words that do not satisfy ϕ. Consequently, the Petri net N satisfies ϕ iff $\mathcal{L}(N) \cap \mathcal{L}(A_{\neg\phi}) = \emptyset$ and $\mathcal{L}(N) \cap \mathcal{L}(B_{\neg\phi}) = \emptyset$. In order to decide these two properties the finite automata are transformed into certain Petri nets and combined with N in a way similar to the classical automata product yielding $N \times N_{A_{\neg\phi}}$ and $N \times N_{B_{\neg\phi}}$ such that $\mathcal{L}(N \times N_{A_{\neg\phi}}) = \mathcal{L}(N) \cap \mathcal{L}(N_{A_{\neg\phi}})$ and $\mathcal{L}(N \times N_{B_{\neg\phi}}) = \mathcal{L}(N) \cap \mathcal{L}(N_{B_{\neg\phi}})$. Esparza then proves that

1. $\mathcal{L}(N) \cap \mathcal{L}(A_{\neg\phi}) \neq \emptyset$ iff the Petri net $N \times N_{A_{\neg\phi}}$ has a reachable dead marking which marks some place of a certain set S.

2. $\mathcal{L}(N) \cap \mathcal{L}(B_{\neg\phi}) \neq \emptyset$ iff the Petri net $N \times N_{B_{\neg\phi}}$ has a run which contains infinitely many occurrences of transitions of a certain set T.

Now the existence of a reachable dead marking in (1.) can be decided solving an exponential number of instances of the reachability problem for Petri nets. The reachability problem is known to be decidable since the early eighties when Kosaraju [96], as well as Mayr [106], presented a non-primitive recursive algorithm. Only recently, Bouziane discovered an EXPSPACE-algorithm [14] which matches the long known complexity lower bound of Lipton [103].

The existence of a run in (2.) containing infinitely many occurrences of transitions of a given set T was shown to be decidable by Jantzen and Valk [83]. Later Yen has shown that this can be decided also within exponential space [150].

Overall, this yields an EXPSPACE-algorithm. Since on the other hand the reachability problem for Petri nets can be encoded in LTL, the model checking problem is also EXPSPACE-hard, proving the problem to be EXPSPACE-complete.

For reactive systems, which often are assumed to provide a certain service forever without ever terminating, another, non-standard semantics is of interest, which only takes infinite runs into account. When restricted to these ω-semantics, the space complexity of Esparza's decision procedure is exponential in the size of the Petri net and double exponential in the size of the formula. Recently, Habermehl [62] has improved on this complexity by showing that the space complexity can be reduced to be polynomial in the size of the formula, which is known to be a lower bound already for finite-state systems. However, also in the ω-setting the problem remains EXPSPACE-hard in the size of the system.

Even more surprisingly, the same complexity is obtained already for the weaker model of BPP and the weaker logic LTL [64]. In this case EXPSPACE-hardness can be shown in five steps:

- First, the EXPSPACE-complete problem whether an exponentially space-bounded Turing-machine accepts a given input, is reduced to the problem whether the exponentially space-bounded universal Turing-machine accepts a given input.

- Second, the universal Turing-machine is encoded in an exponentially space-bounded universal Minsky n-counter machine, for which, consequently, the acceptance problem is EXPSPACE-hard.
- Third, this universal Minsky n-counter machine can be encoded as a parallel composition of a BPP and a finite automaton yielding a PPDA similar to the construction in [103] or [53]. The size of this finite automaton is a function in the size of the finite control of the universal counter machine and thus fixed and finite.
- Fourth, the problem if the resulting PPDA has an infinite run can be encoded in a model checking problem for BPP and an LTL-formula. The LTL-formula depends only on the fixed finite automaton and is thus also fixed. This encoding of the automaton in LTL is possible, since, by choosing the atomic actions accordingly, the automaton can be made star-free.

Summarizing, we have the following complexities when model checking LTL, respectively LTμ.

LTL and LTμ with standard semantics		
	General	Fixed Formula
BPP / PN	EXPSPACE-complete	EXPSPACE-complete

LTL and LTμ with ω-semantics		
	Fixed System	Fixed Formula
BPP / PN	PSPACE-complete	EXPSPACE-complete

The Logic CLTL Pursuing their research on CLTL, Bouajjani and Habermehl have shown that $\text{CLTL}_{\Box}$ is also decidable for Petri nets [12]. Since Petri nets are in contrast to pushdown processes not semilinear, the proof for the decidability of $\text{CLTL}_{\Box}$ on pushdown processes does not carry over. This time satisfiability of $\text{CLTL}_{\Diamond}$ is reduced in several steps to the reachability problem in Petri nets.

- First, the regular part of the $\text{CLTL}_{\Diamond}$ formula ϕ is transformed into an equivalent Büchi automaton $\mathcal{B}$ which is then combined with the net N such that $\mathsf{N} \times \mathcal{B}$ represents their synchronized product.
- Second, one can show that the set of markings M of $\mathsf{N} \times \mathcal{B}$ from which the Büchi automaton accepts every run is semilinear.
- Third, since the counting constraint part of ϕ also describes a semilinear set of markings M', and since semilinear sets are closed under intersection, the satisfiability problem reduces to the reachability problem for the semilinear set of markings $\mathsf{M} \cap \mathsf{M}'$.
- Fourth and last, the previous reachability problem for semilinear sets can be reduced to the classical reachability problem for single markings in Petri nets.

3.3.3 PA

LTL and LTμ Although LTL is decidable for BPA, as well as for BPP, this result does not longer hold in their common generalization PA. As shown in [13], a Minsky machine can be simulated by three BPA processes in parallel, such that their "honest" synchronization, as well as the reachability of the halting state, is expressible by a LTL formula. More technically, the halting of the Minsky machine is encoded as the nonemptiness of the intersection of an ω-star-free language with a PA ω-language. Undecidability of the model checking problem for LTL and PA follows then immediately from the undecidability of the halting problem for Minsky machines [117].

The Logic CLTL As a corollary, the undecidability of model checking LTL also proves that PA ω-languages are not closed under intersection with ω-star-free languages. On the other hand, it is shown in [12] that PA ω-languages are closed under intersection with the smaller class of simple ω-regular languages, i.e., ω-languages definable by a finite-state Büchi automaton where every loop in its transition graph is a self-loop. Since the fragment simple-PLTL$_\Box$ of CLTL, which is incomparable to LTL, expresses exactly the simple ω-regular languages, it can, finally, be shown that simple-PLTL$_\Box$ is indeed decidable for PA.

3.4 Summary

From a bird's perspective we may conclude that sequential-type systems, like BPA and PDA, are inherently easier to analyse than parallel-type systems. This conclusion is supported by both the (un)decidability as well as the complexity results:

For BPA and PDA, most branching and linear-time logics are decidable, while for parallel-type systems, like BPP and Petri nets, except for the very restrictive branching-time logics HML and EF only linear-time logics are decidable. In particular the modal μ-calculus as well as LTL (and even CLTL$_\Box$) are decidable for PDA, while on the other hand all branching-time logics except HML are undecidable for Petri nets, and merely the linear-time μ-calculus with true as only atomic proposition is decidable for Petri nets.

Considering the complexity results for model checking the picture is similar. For example on the sequential side model checking the modal μ-calculus and the linear-time μ-calculus are both EXPTIME-complete for PDA, while on the parallel side the model checking problem for Petri nets with respect to the linear-time μ-calculus is already EXPSPACE-complete. The following corresponding tables summarize these results and the pointers to the relevant literature. To abbreviate the notation, G: indicates the complexity of the general problem, while F: specifies the complexity of the problem when the formula is fixed.

The central results presented in this chapter concern worst-case estimations within the classical complexity class structure. This classification, although constituting a very good guideline also for practical purposes, must often be re-

Class	Logic	Complexity
BPA	HML	G: PSPACE-complete [112] F: PTIME
BPA	modal μ-calculus	G: EXPTIME-complete [149, 112] F: PTIME [29]
PDA	modal μ-calculus	G: EXPTIME-complete [149] F: EXPTIME-complete [149]
BPP	EF	G: PSPACE-complete [52, 109, 111] F: Σ_k^P [52, 109, 111]
PA	EF	G: PSPACE-hard [110, 111] F: Σ_k^P [111]

Class	Logic	Complexity
FSA	LTL / LTμ	G: PSPACE-complete [135, 144] F: PTIME
BPA	LTL / LTμ	G: EXPTIME-complete [111] F: PTIME [11]
PDA	LTL / LTμ	G: EXPTIME-complete [11] F: PTIME [11]
BPP / PN	LTL / LTμ	G: EXPSPACE-complete [51, 111] F: EXPSPACE-complete [51, 111]

Figure 6: Complexities of model checking problems.

fined in order to provide a real basis for implementation decisions. We therefore propose to investigate

- finer complexity class structures, e.g., in many cases people only accept linear or at most quadratic algorithms, and even constant factors may play a significant role;
- average or 'in practice' behaviours: systems developed by people are often much better behaved as suggested by worst-case analyses; and
- special restrictive but practical system/property combinations, which can be treated efficiently.

Whereas the first two points do not require any further explanation, the third should become clearer by considering program analysis as an illustrating example. Here the so-called *bitvector problems* constitute a highly relevant but tractable class. In fact, along the lines of [110, 57, 131] it is straightforward to deduce that interprocedural bitvector analyses, even for programs with fork/join parallelism and shared variables, admit model checking in linear time. Special cases like this, which depend on rather specific patterns of formulae and/or models, will always have to be considered case by case. However, we are convinced that the overview of the general complexities provides beneficial guidelines even

for such refined complexity considerations, as one cannot construct a scenario avoiding 'bottlenecks' without knowing exactly where they are located.

References

[1] P.A. Abdulla and K. Čerāns. Simulation is decidable for one-counter nets. In D. Sangiorgi and R. de Simone, editors, *CONCUR'98*, LNCS 1466, pages 253–268, Nice, 1998. Springer.

[2] R. Alur, C. Courcoubetis, T.A. Henzinger, N. Halbwachs, P.H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.

[3] R. Alur and T.A. Henzinger. Real-time systems = discrete systems + clock variables. *Software Tools for Technology Transfer*, 1:86–109, 1997.

[4] J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. Decidability of bisimulation equivalence for processes generating context-free languages. In J.W. de Bakker, A.J. Nijman, and P.C. Treleaven, editors, *PARLE'87*, LNCS 259, pages 94–113, Eindhoven, 1987. Springer.

[5] J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. Decidability of bisimulation equivalence for processes generating context-free languages. *Journal of the ACM*, 40(3):653–682, 1993.

[6] Y. Bar-Hillel, M. Perles, and E. Shamir. On formal properties of simple phrase structure grammars. *Zeitschrift für Phonetik, Sprachwissenschaft, und Kommunikationsforschung*, 14:143–177, 1961.

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

[8] C. Berge. *Graphs and Hypergraphs.* North-Holland, Amsterdam, 1973.

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

[10] A. Bouajjani, R. Echahed, and R. Robbana. Verification of nonregular temporal properties for context-free processes. In B. Jonsson and J. Parrow, editors, *CONCUR'94*, LNCS 836, pages 81–97, Uppsala, 1994. Springer.

[11] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In A. Mazurkiewicz and J. Winkowski, editors, *CONCUR'97*, LNCS 1243, pages 135–150, Warsaw, 1997. Springer.

[12] A. Bouajjani and P. Habermehl. Constrained properties, semilinear systems, and Petri nets. In U. Montanari and V. Sassone, editors, *CONCUR'96*, LNCS 1119, pages 481–497, Pisa, 1996. Springer.

[13] R. Bouajjani, R. Echahed, and P. Habermehl. On the verification of nonregular properties for nonregular processes. In *LICS'95*, pages 123–133, San Diego, 1995. IEEE Computer Society Press.

[14] Z. Bouziane. A primitive recursive algorithm for the general Petri net reachability problem. In *FOCS'98*, pages 130–136, Palo Alto, 1998. IEEE Computer Society Press.

[15] J.C. Bradfield. *Verifying Temporal Properties of Systems.* Birkhäuser, 1992.

[16] J.C. Bradfield. The modal mu-calculus alternation hierarchy is strict. In U. Montanari and V. Sassone, editors, *CONCUR'96*, LNCS 1119, pages 233–246, Pisa, 1996. Springer.

[17] J.C. Bradfield and C.P. Stirling. Modal logics for processes. Chapter 1.4 of this volume.

[18] R. Büchi. On a decision method in restricted second order arithmetic. In *Proceedings of the 1960 International Congress for Logic, Methodology, and Philosophy of Science*, pages 1–11, Berkeley, 1962. Stanford University Press.

[19] R. Büchi. Regular canonical systems. *Archive für Mathematische Logik und Grundlagenforschung*, 6:91–111, 1964.

[20] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking 10^{20} states and beyond. In *LICS'90*, pages 428–439, Philadelphia, 1990. IEEE Computer Society Press.

[21] O. Burkart. *Automatic Verification of Sequential Infinite-State Processes.* LNCS 1354. Springer, 1997.

[22] O. Burkart. Model-checking rationally restricted right closures of recognizable graphs. *Electronic Notes in Theoretical Computer Science*, 9, 1997.

[23] O. Burkart, D. Caucal, and B. Steffen. An elementary bisimulation decision procedure for arbitrary context-free processes. In J. Wiedermann and P. Hájek, editors, *MFCS'95*, LNCS 969, pages 423–433, Prague, 1995. Springer.

[24] O. Burkart, D. Caucal, and B. Steffen. Bisimulation collapse and the process taxonomy. In U. Montanari and V. Sassone, editors, *CONCUR'96*, LNCS 1119, pages 247–262, Pisa, 1996. Springer.

[25] O. Burkart and J. Esparza. More infinite results. *Electronic Notes in Theoretical Computer Science*, 5, 1997.

[26] O. Burkart and Y.-M. Quemener. Model-checking of infinite graphs defined by graph grammars. *Electronic Notes in Theoretical Computer Science*, 5, 1997.

[27] O. Burkart and B. Steffen. Model checking for context-free processes. In W.R. Cleaveland, editor, *CONCUR'92*, LNCS 630, pages 123–137, Stony Brook, 1992. Springer.

[28] O. Burkart and B. Steffen. Composition, decomposition and model-checking of pushdown processes. *Nordic Journal of Computing*, 2:89–125, 1995.

[29] O. Burkart and B. Steffen. Model-checking the full-modal mu-calculus for infinite sequential processes. In P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, editors, *ICALP'97*, LNCS 1256, pages 419–429, Bologna, 1997. Springer.

[30] D. Caucal. A fast algorithm to decide on the equivalence of stateless dpda. *RAIRO*, 27(1):23–48, 1990.

[31] D. Caucal. Graphes canoniques de graphes algébriques. *RAIRO*, 24(4):339–352, 1990.

[32] D. Caucal. On the regular structure of prefix rewriting. *Theoretical Computer Science*, 10:61–86, 1992.

[33] D. Caucal. On infinite transition graphs having a decidable monadic theory. In F. Meyer auf der Heide and B. Monien, editors, *ICALP'96*, LNCS 1099, pages 194–205, Paderborn, 1996. Springer.

[34] D. Caucal and R. Monfort. On the transition graphs of automata and grammars. In R.H. Möhring, editor, *Graph-Theoretic Concepts in Computer Science*, LNCS 484, pages 311–337, Berlin, 1990. Springer.

[35] S. Christensen. *Decidability and Decomposition in Process Algebras*. PhD thesis, The University of Edinburgh, Department of Computer Science, ECS-LFCS-93-278, 1993.

[36] S. Christensen, Y. Hirshfeld, and F. Moller. Bisimulation equivalence is decidable for basic parallel processes. In E. Best, editor, *CONCUR'93*, LNCS 715, pages 143–157, Hildesheim, 1993. Springer.

[37] S. Christensen, Y. Hirshfeld, and F. Moller. Decomposability, decidability and axiomatisability for bisimulation equivalence on basic parallel processes. In *LICS'93*, pages 386–396, Montreal, 1993. IEEE Computer Society Press.

[38] S. Christensen, H. Hüttel, and C. Stirling. Bisimulation equivalence is decidable for all context-free processes. In W.R. Cleaveland, editor, *CONCUR'92*, LNCS 630, pages 138–147, Stony Brook, 1992. Springer.

[39] S. Christensen, H. Hüttel, and C. Stirling. Bisimulation equivalence is decidable for all context-free processes. *Information and Computation*, 12(2):143–148, 1995.

[40] E.M. Clark, E.A. Emerson, and A.P. Sistla. Automatic verification of finite state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[41] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, 1999.

[42] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proceedings of the Workshop on Logics of Programs*, LNCS 131, pages 52–71, Yorktown Heights, 1981. Springer.

[43] R. Cleaveland, M. Klein, and B. Steffen. Faster model checking for the modal mu-calculus. In G. Bochmann and D.K. Probst, editors, *CAV'92*, LNCS 663, pages 410–422, Montreal, 1992. Springer.

[44] R. Cleaveland and Sokolsky. O. Equivalence and preorder algorithms for finite-state systems. Chapter 2.2 of this volume.

[45] R. Cleaveland and B. Steffen. Computing behavioural relations, logically. In J Leach Albert, B. Monien, and Rodríguez, editors, *ICALP'91*, LNCS 510, pages 127–138, Madrid, 1991. Springer.

[46] 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.

[47] M. Dam. Fixpoints of Büchi automata. In R. Shyamasundar, editor, *FSTTCS'92*, LNCS 652, pages 39–50, New Delhi, 1992. Springer.

[48] L.E. Dickson. Finiteness of the odd perfect and primitive abundant numbers with distinct factors. *American Journal of Mathematics*, 35:413–422, 1985.

[49] E.A. Emerson and C.-L. Lei. Efficient model checking in fragments of the propositional mu-calculus. In *LICS'86*, pages 267–278, Cambridge, Massachusetts, 1986. IEEE Computer Society Press.

[50] E.A. Emerson and Srinivasan. Branching time temporal logic. In J.W. de Bakker, W.-P. de ROever, and G. Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, LNCS 354, pages 123–172. Springer, 1989.

[51] J. Esparza. On the decidability of model checking for several μ-calculi and Petri nets. In S. Tison, editor, *CAAP'94*, LNCS 787, pages 115–129, Edinburgh, 1994. Springer.

[52] J. Esparza. Decidability of model-checking for concurrent infinite-state systems. *Acta Informatica*, 34:85–107, 1997.

[53] J. Esparza. Decidability and complexity of Petri net problems: An introduction. In W. Reisig and G. Rozenberg, editors, *Lectures on Petri Nets I: Basic Models*, LNCS 1491, pages 374–428. Springer, 1998.

[54] J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. Efficient algorithms for model checking pushdown systems. Submitted for publication.

[55] J. Esparza, P. Jančar, and F. Moller. Petri nets and regular behaviours. *Journal of Computer and System Sciences*, 59(3):476–503, 1999.

[56] J. Esparza and A. Kiehn. On the model checking problem for branching time logics and basic parallel processes. In P. Wolper, editor, *CAV'95*, LNCS 939, pages 353–366, Liege, 1995. Springer.

[57] J. Esparza and A. Podelski. Efficient algorithms for pre* and post* in interprocedural parallel flow graphs. In *POPL'00*, pages 1–12, Boston, 2000. ACM Press.

[58] R.W. Floyd. Assigning meanings to programs. In J.T. Schwartz, editor, *Symposium on Applied Mathematics*, volume 19, pages 19–32. American Mathematical Society, 1967.

[59] R.J. van Glabbeek. The linear time - branching time spectrum. In J.C.M Baeten and J.W. Klop, editors, *CONCUR'90*, LNCS 458, pages 278–297, Amsterdam, 1990. Springer.

[60] J.F. Groote. A short proof of the decidability of bisimulation for normed BPA-processes. *Information Processing Letters*, 42:167–171, 1991.

[61] J.F. Groote and H. Hüttel. Undecidable equivalences for basic process algebra. *Information and Computation*, 11(2):354–371, 1994.

[62] P. Habermehl. On the complexity of the linear time mu-calculus for Petri nets. In P. Azéma and G. Balbo, editors, *18th International Conference on Application and Theory of Petri Nets*, LNCS 1248, pages 102–116, Toulouse, 1997. Springer.

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

[64] P. Hebermehl and R. Mayr. Personal communications. Jan 2000.

[65] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32(1):137–161, 1985.

[66] Y. Hirshfeld. Petri nets and the equivalence problem. In E. Börger, Y. Gurevich, and K. Meinke, editors, *CSL'93*, LNCS 832, pages 165–174, Swanseea, 1993. Springer.

[67] Y. Hirshfeld. Congruences in commutative semigroups. Technical Report ECS-LFCS-94-291, Department of Computer Science, University of Edinburgh, UK, 1994.

[68] Y. Hirshfeld and M. Jerrum. Bisimulation equivalence is decidable for normed Process Algebra. In J Wiedermann, P. van Emde Boas, and M. Nielsen, editors, *ICALP'99*, LNCS 1644, pages 412–421, Prague, 1999. Springer.

[69] Y. Hirshfeld, M. Jerrum, and F. Moller. A polynomial algorithm for deciding bisimilarity of normed context-free processes. In *FOCS'94*, pages 623–631, Santa Fe, 1994. IEEE Computer Society Press.

[70] Y. Hirshfeld, M. Jerrum, and F. Moller. A polynomial algorithm for deciding bisimilarity of normed context-free processes. *Theoretical Computer Science*, 15:143–159, 1996.

[71] Y. Hirshfeld, M. Jerrum, and F. Moller. A polynomial algorithm for deciding bisimulation equivalence of normed basic parallel processes. *Mathematical Structures in Computer Science*, 6:251–259, 1996.

[72] Y. Hirshfeld and F. Moller. A fast algorithm for deciding bisimilarity of normed context-free processes. In B. Jonsson and J. Parrow, editors, *CONCUR'94*, LNCS 836, pages 48–63, Uppsala, 1994. Springer.

[73] G.J. Holzmann. *Design and Validation of Computer Programs.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1991.

[74] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison Wesley, 1979.

[75] H. Hungar. Local model checking for parallel compositions of context-free processes. In B. Jonsson and J. Parrow, editors, *CONCUR'94*, LNCS 836, pages 114–128, Uppsala, 1994. Springer.

[76] H. Hungar. Model-checking of macro processes. In D.L. Dill, editor, *CAV'94*, LNCS 818, pages 169–181, Stanford, 1994. Springer.

[77] H. Hungar. Model-checking and higher-order recursion. In L. Pacholski, editor, *MFCS'99*, LNCS 1672, pages 149–159, Szklarska Poreba, 1999. Springer.

[78] H. Hungar and B. Steffen. Local model-checking for context-free processes. *Nordic Journal of Computing*, 1(3):364–385, 1994.

[79] H. Hüttel. Undecidable equivalences for basic parallel processes. In M. Hagiya and J.C. Mitchell, editors, *TACS'94*, LNCS 789, pages 454–464, Sendai, 1994. Springer.

[80] H. Hüttel and C. Stirling. Actions speak louder than words: Proving bisimilarity for context-free processes. In *LICS'91*, pages 376–386, Amsterdam, 1991. IEEE Computer Society Press.

[81] D.T. Huynh and L. Tian. Deciding bisimilarity of normed context-free processes is in Σ_2^{P}. *Theoretical Computer Science*, 12:183–197, 1994.

[82] D.T. Huynh and L. Tian. On deciding readiness and failure equivalences for processes. *Information and Computation*, 11:193–205, 1995.

[83] M. Jantzen and R. Valk. The residue of vector sets with applications to decidability problems in Petri nets. *Acta Informatica*, 21:643–674, 1985.

[84] P. Jančar. Decidability questions for bisimilarity of Petri nets and some related problems. In P. Enjalbert, E.W. Mayr, and K.W. Wagner, editors, *STACS'94*, LNCS 775, pages 581–592, Caen, 1994. Springer.

[85] P. Jančar. Undecidability of bisimilarity for Petri nets and some related problems. *Theoretical Computer Science*, 148:281–301, 1995.

[86] P. Jančar. Bisimulation equivalence is decidable for one-counter processes. In P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, editors, *ICALP'97*, LNCS 1256, pages 549–559, Bologna, 1997. Springer.

[87] P. Jančar, A. Kučera, and R. Mayr. Deciding bisimulation-like equivalences with finite-state processes. In K.G. Larsen, S. Skyum, and G. Winskel, editors, *ICALP'98*, LNCS 1443, pages 200–211, Aalborg, 1998. Springer.

[88] P. Jančar, A. Kučera, and F. Moller. Simulation and bisimulation over one-counter processes. In H. Reichel and S. Tison, editors, *STACS'2000*, LNCS 1770. Springer, 2000.

[89] P. Jančar and F. Moller. Checking regular properties of Petri nets. In I. Lee and S.A. Smolka, editors, *CONCUR'95*, LNCS 962, pages 348–362, Philadelphia, 1995. Springer.

[90] P. Jančar and F. Moller. Simulation of one-counter nets via colouring (abstract). In A. Finkel, editor, *Workshop Journées Systèmes Infinis*, pages 1–6, Cachan, 1998. Report CNRS URA 2236, Ecole Normale Supérieure de Cachan. Full 10-page paper available as Uppsala University CSD Report No. 159 at http://www.csd.uu.se/papers/report.html.

[91] P. Jančar and F. Moller. Techniques for decidability and undecidability for bisimilarity. In J.C.M. Baeten and S. Mauw, editors, *CONCUR'99*, LNCS 1664, pages 30–45, Eindhoven, 1999. Springer.

[92] P. Jančar, F. Moller, and Z. Sawa. Simulation problems for one-counter machines. In J. Pavelka, G. Tel, and M. Bartosek, editors, *SOFSEM'99*, LNCS 1725, pages 398–407, Milovy, 1999. Springer.

[93] D. Johnson. A catalog of complexity classes. In J van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, chapter 2, pages 67–161. Elsevier, 1990.

[94] D.E. Knuth, J.H. Morris, and V.R. Pratt. Fast pattern matching in strings. *SIAM Journal of Computing*, 6:323–350, 1977.

[95] A.J. Korenjak and J.E. Hopcroft. Simple deterministic languages. In *7th Annual IEEE Symposium on Switching and Automata Theory*, pages 36–46, Berkeley, 1966. IEEE.

[96] S.R. Kosaraju. Decidability of reachability in vector addition systems. In *STOC'82*, pages 267–281, San Fransisco, 1982.

[97] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[98] A. Kučera and R. Mayr. Simulation preorder on simple process algebras. In J Wiedermann, P. van Emde Boas, and M. Nielsen, editors, *ICALP'99*, LNCS 1644, pages 503–512, Prague, 1999. Springer.

[99] A. Kučera and R. Mayr. Weak bisimilarity with infinite state systems can be decided in polynomial time. In J.C.M. Baeten and S. Mauw, editors, *CONCUR'99*, LNCS 1664, pages 368–382, Eindhoven, 1999. Springer.

[100] L. Lamport. Verification and specification of concurrent programs. In J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *A Decade of Concurrency*, LNCS 803, pages 347–374, Noordwijkerhout, 1994. Springer.

[101] G. Lenzi. A hierarchy theorem for the mu-calculus. In F. Meyer auf der Heide and B. Monien, editors, *ICALP'96*, LNCS 1099, pages 87–109, Paderborn, 1996. Springer.

[102] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *POPL'85*, pages 97–107, New Orleans, 1985. ACM Press.

[103] R. Lipton. The reachability problem requires exponential space. Technical Report 62, Yale University, 1976.

[104] D. Lugiez and Ph. Schnoebelen. The regular viewpoint on PA-processes. In D. Sangiorgi and R. de Simone, editors, *CONCUR'98*, LNCS 1466, pages 50–66, Nice, 1998. Springer.

[105] Z. Manna and A. Pnueli. *Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer, 1992.

[106] E.W. Mayr. An algorithm for the general Petri net reachability problem. *SIAM Journal of Computing*, 13(3):441–459, 1984.

[107] R. Mayr. On the complexity of bisimulation problems for basic parallel processes. Submitted for publication.

[108] R. Mayr. On the complexity of bisimulation problems for pushdown automata. Submitted for publication.

[109] R. Mayr. Weak bisimulation and model checking for basic parallel processes. In V. Chandru and V. Vinay, editors, *FSTTCS'96*, LNCS 1180, pages 88–99, Hyderabad, 1996. Springer.

[110] R. Mayr. Model checking PA-processes. In A. Mazurkiewicz and J. Winkowski, editors, *CONCUR'97*, LNCS 1243, pages 332–346, Warsaw, 1997. Springer.

[111] R. Mayr. *Decidability and Complexity of Model Checking Problems for Infinite-State Systems.* PhD thesis, Technische Universität München, 1998.

[112] R. Mayr. Strict lower bounds for model checking BPA. *Electronic Notes in Theoretical Computer Science*, 18, 1998.

[113] R. Milner. *A Calculus of Communicating Systems.* LNCS 92. Springer, 1980.

[114] R. Milner. A complete inference system for a class of regular behaviours. *Journal of Computer and System Sciences*, 28:439–466, 1984.

[115] R. Milner. *Communication and Concurrency.* Prentice-Hall, 1989.

[116] R. Milner and F. Moller. Unique decomposition of processes. *Theoretical Computer Science*, 107:357–363, 1993.

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

[118] F. Moller. Infinite results. In U. Montanari and V. Sassone, editors, *CONCUR'96*, LNCS 1119, pages 195–216, Pisa, 1996. Springer.

[119] F. Moller. A taxonomy of infinite state processes. *Electronic Notes in Theoretical Computer Science*, 18, 1998.

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

[121] E.F. Moore. Gedanken experiments on sequential machines. *Automata Studies*, pages 129–153, 1956.

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

[123] D. Niwiński. On fixed-point clones. In L. Kott, editor, *ICALP'86*, LNCS 226, pages 464–473, Rennes, 1986. Springer.

[124] D. Niwiński. Fixed point characterization of infinite behaviour of finite state systems. *Theoretical Computer Science*, 189:1–69, 1997.

[125] D. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *5th GI Conference on Theoretical Computer Science*, LNCS 104, pages 167–183, Karlsruhe, 1981. Springer.

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

[127] A. Pnueli. The temporal logic of programs. In *FOCS'77*, pages 46–57, Providence, 1977. IEEE Computer Society Press.

[128] R.O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the AMS*, 141:1–35, 1969.

[129] L. Redei. *The Theory of Finitely Generated Commutative Semigroups*. Oxford University Press, 1965.

[130] A. Salomaa. Two complete axiom systems for the algebra of regular events. *Journal of the ACM*, 13:158–169, 1966.

[131] H. Seidel and B. Steffen. Constraint-bssed inter-procedural analysis of parallel programs. In G. Smolka, editor, *ESOP'00*, LNCS 1782, pages 135–150, Berlin, 2000. Springer.

[132] G. Sénizergues. The equivalence problem for deterministic pushdown automata is decidable. In P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, editors, *ICALP'97*, LNCS 1256, pages 671–681, Bologna, 1997. Springer.

[133] G. Sénizergues. L(A) = L(B) ? Technical Report 1161-97, LaBRI, Bordeaux, France, 1997.

[134] G. Sénizergues. Decidability of bisimulation equivalence for equational graphs of finite out-degree. In *FOCS'98*, pages 120–129, Palo Alto, 1998. IEEE Computer Society Press.

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

[136] C. Stirling. Local model checking games. In I. Lee and S.A. Smolka, editors, *CONCUR'95*, LNCS 962, pages 1–11, Philadelphia, 1995. Springer.

[137] C. Stirling. Decidability of bisimulation equivalence for normed pushdown processes. In U. Montanari and V. Sassone, editors, *CONCUR'96*, LNCS 1119, pages 217–232, Pisa, 1996. Springer.

[138] C. Stirling. Decidability of dpda equivalence. Technical Report ECS-LFCS-99-411, Department of Computer Science, University of Edinburgh, UK, 1999.

[139] J. Stříbrná. Hardness results for weak bisimilarity of simple process algebras. *Electronic Notes in Theoretical Computer Science*, 18, 1998.

[140] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

[141] D. Taubner. *Finite Representations of CCS and TCSP Programs by Automata and Petri Nets*. LNCS 369. Springer, 1989.

[142] W. Thomas. A combinatorial approach to the theory of ω-automata. *Information and Control*, 48:261–283, 1981.

[143] L.G. Valiant and M.S. Paterson. Deterministic one-counter automata. *Journal of Computer and System Sciences*, 10:340–350, 1975.

[144] M.Y. Vardi. A temporal fixpoint calculus. In *POPL'88*, pages 250–259, San Diego, 1988. ACM Press.

[145] M.Y. Vardi. Alternating automata and program verification. In J. van Leeuwen, editor, *Computer Science Today: Recent Trends and Developments*, LNCS 1000, pages 471–485. Springer, 1995.

[146] M.Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *Journal of Computer and System Sciences*, 32:183–221, 1986.

[147] K. Varpaaniemi. Prod 3.3.02: An advanced tool for efficient reachability analysis. Technical report, Department of Computer Science and Engineering, Helsinki University of Technology, Finland, 1998. Available at `http://www.tcs.hut.fi/pub/prod`.

[148] F Wallner. Model checking LTL using net unfoldings. In A.J. Hu and M.Y. Vardi, editors, *CAV'98*, LNCS 1427, pages 207–218, Vancouver, 1998. Springer.

[149] I. Walukiewicz. Pushdown processes: Games and model-checking. In R. Alur and T.A. Henzinger, editors, *CAV'96*, LNCS 1102, pages 62–74, New Brunswick, 1996. Springer.

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