Invited paper for CONCUR'96, *Lecture Notes in Computer Science* **1119**:195-216 (1996).

Infinite Results*

Faron Moller
Computing Science Department
Uppsala University

P.O. Box 311
S-751 05 Uppsala, SWEDEN

Abstract

Recently there has been a spurt of activity in concurrency theory centred on the analysis of infinite-state systems. Much of this work stems from a task dedicated to the study in the recently-concluded ESPRIT BRA Concur2, and much of it has subsequently appeared in the proceedings of the annual CONCUR conference. In this paper, we present an overview of various results obtained regarding expressivity, decidability, and complexity, focussing on the various techniques exploited in each case.

1 Introduction

The study of (sequential) program verification has an inherent theoretical barrier in the form 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 infinite number of states, and knowing if the execution of the program will lead to any particular state, such as the halting state, will typically be impossible. However, this has not prevented a 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 exploring the reachable states of any given system, for instance with the goal of detecting whether or not a halting, ie, deadlocked, state is accessible. This abstraction has been warranted up

*Research partially supported by TFR grant 221-95-244: *Verification via Decomposition*.

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 depend 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 [5]. However, such approaches are inherently bound to the analysis of finite-state systems.

Recently, interest in addressing the problem of analysing infinite-state spaces 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 be introduced to successfully analyse systems expressed in such formalisms.

A particularly relevant question here then is deciding when two infinite-state systems are in some semantic sense equal. Such questions are of course not new in the field of theoretical computer science. Since the proof by Moore [42] 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 [3] were the first to demonstrate in 1961 that the class of languages defined by context-free grammars was too wide to permit a decidable theory for language equivalence. The search for a more precise dividing line is still active, with the most outstanding open problem concerning the decidability of language equivalence between deterministic push-down automata.

Decidability questions for Petri nets were addressed already two decades ago, with the thesis of Hack [19]. 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 [36] Milner exploits the relationship between regular (finite-state) automata as discussed in [46] and regular behaviours to present the decidability and a complete axiomatisation of bisimulation equivalence for finite-state behaviours, whilst in his textbook [38] 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 [49]. These results are as expected; however, real interest was stirred with the discovery by Baeten, Bergstra and Klop [1, 2] that bisimulation equivalence is decidable for a family of infinite-state automata generated by a general class of context-free grammars.

Much of the effort on exploring such decidability issues has taken place within the framework of the recently-concluded ESPRIT BRA Concur2 which included a task dedicated to the study of infinite states. As such, many of the milestones in the field have appeared in the proceedings of the annual CONCUR conferences, as well as in the proceedings of the satellite workshop INFINITY which accompanies the CONCUR'96 conference. In this paper, we present an overview of various results obtained regarding expressivity, decidability, and complexity, with particular emphasis on the bisimulation equivalence problem, focussing on the various techniques exploited in each case.

2 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 2.1 A *labelled transition system* is a tuple $\langle S, \Sigma, \longrightarrow, \alpha_0, F\rangle$ where

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

This notion of a labelled transition system differs from the standard definition of a finite state automata (as for example given in [27]) 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 2.3 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 2.3 based on PDA which accept by final state, but we do not pursue this alternative in this paper.)

In this overview, we follow the example set by Caucal [9] 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 2.2 A *sequential labelled rewrite transition system* is a tuple $\langle V, \Sigma, P, \alpha_0, F\rangle$ 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 catenation, which is thus interpretted as parallel, rather than sequential, composition.

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 by Caucal, 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. (Type 1—context-sensitive—rewrite systems do not feature in this hierarchy since the rewrite rules by definition are only applied to the prefix of a composition.) This hierarchy provides an elegant classification of several important classes of transition systems which have been defined and studied independent of their appearance as particular rewrite systems. This classification is presented as follows.

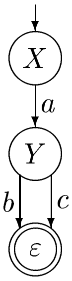
	Restriction on the rules $\alpha \xrightarrow{a} \beta$ of P	Restriction on F	Sequential composition	Parallel composition
Type 0:	*none*	*none*	PDA	PN
Type $1\frac{1}{2}$:	$\alpha \in Q\Gamma$ and $\beta \in Q\Gamma^*$ where $V = Q \uplus \Gamma$	$F = Q$	PDA	PPDA
Type 2:	$\alpha \in V$	$F = \{\varepsilon\}$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

In the remainder of this section, we explain the classes of transition systems which are represented in this table, working upwards starting with the most restrictive class.

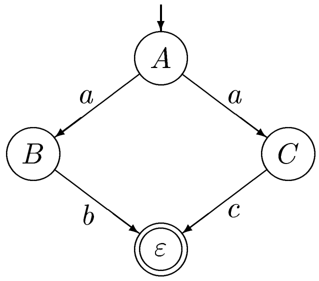
FSA represents the class of finite-state automata. Clearly if the rules are restricted to be of the form $A \xrightarrow{a} B$ or $A \xrightarrow{a} \varepsilon$ with $A, B \in V$, then the reachable states of both the sequential and parallel transition systems will be a subset of the finite set of variables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

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.*

$$X \xrightarrow{a} Y \qquad Y \xrightarrow{b} \varepsilon \qquad Y \xrightarrow{c} \varepsilon$$



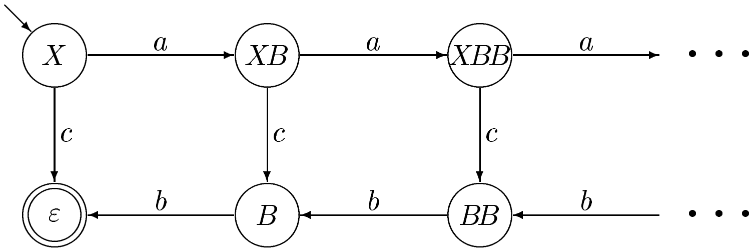
$$A \xrightarrow{a} B \qquad A \xrightarrow{a} C \qquad B \xrightarrow{b} \varepsilon \qquad C \xrightarrow{c} \varepsilon$$



BPA represents the class of Basic Process Algebra processes of Bergstra and Klop [4], 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.*

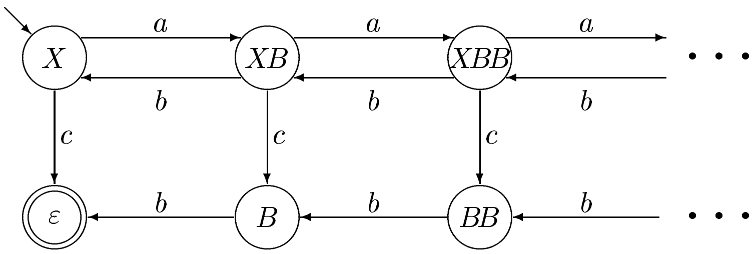
$$X \xrightarrow{a} XB \qquad X \xrightarrow{c} \varepsilon \qquad B \xrightarrow{b} \varepsilon$$



BPP represents the class of Basic Parallel Processes introduced by Christensen [11] 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.*

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



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 [9] 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 a rewrite system, take n to be at least as large as the length of any sequence appearing on the left hand side of any of its rules, and strictly larger than the length of any final state. Let $Q = \{ p_\alpha : \alpha \in V^* \text{ and length}(\alpha) < n \}$ and $\Gamma = V \cup \{ Z_\alpha : \alpha \in V^* \text{ and length}(\alpha) \leq n \}$. Every final transition state α 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 $\alpha\beta\gamma$ with length$(\alpha) < n$, length$(\beta\gamma) > 0$ only if length$(\alpha) = n-1$, and length$(\beta) > 0$ only if length$(\gamma) = n$, is represented in the PDA by $p_\alpha \beta Z_\gamma$, that is, by the PDA being in control state p_α with the sequence βZ_γ 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 2.3 *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 $p_X Z_\varepsilon$. (We omit rules corresponding to the unreachable states.)

$$\begin{array}{lll}
p_X Z_\varepsilon \xrightarrow{a} p_X Z_B & p_X Z_{BB} \xrightarrow{a} p_X B Z_{BB} & p_B Z_\varepsilon \xrightarrow{b} p_\varepsilon \\
p_X Z_\varepsilon \xrightarrow{c} p_\varepsilon & p_X Z_{BB} \xrightarrow{b} p_X Z_B & p_B Z_B \xrightarrow{b} p_B Z_\varepsilon \\
 & p_X Z_{BB} \xrightarrow{c} p_B Z_B & p_B Z_{BB} \xrightarrow{b} p_B Z_B \\
 & & p_B B \xrightarrow{b} p_B \\
p_X Z_B \xrightarrow{a} p_X Z_{BB} & p_X B \xrightarrow{a} p_X BB & \\
p_X Z_B \xrightarrow{b} p_X Z_\varepsilon & p_X B \xrightarrow{b} p_X & \\
p_X Z_B \xrightarrow{c} p_B Z_\varepsilon & p_X B \xrightarrow{c} p_B &
\end{array}$$

This can be expressed more simply by the following PDA with initial state pZ.

$$\begin{array}{lll}
pZ \xrightarrow{a} pBZ & pB \xrightarrow{a} pBB & qZ \xrightarrow{c} q \\
pZ \xrightarrow{c} q & pB \xrightarrow{b} p & qB \xrightarrow{b} q \\
 & pB \xrightarrow{c} pBB &
\end{array}$$

Note that, as is reflected in the above construction, every BPA is given by a single-state PDA; the reverse identification is also immediately evident. However, we shall see in

Section 2.2 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.

PPDA represents the class of 'parallel' push-down automata, which 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 PPDA with initial state pX.*

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

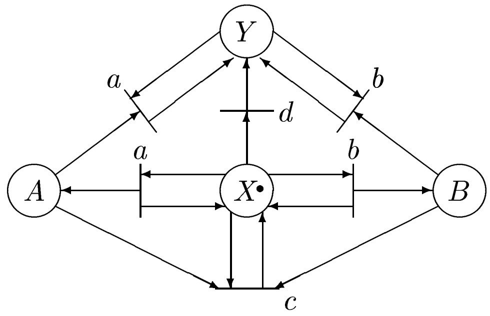
Note that when the stack alphabet has only one element, PDA and PPDA trivially coincide. Also note that BPP coincides with the class of single-state PPDA. We shall see in Section 2.2 that any PPDA presentation of the transition system of Example 1 must have at least 2 control states: this transition system is not represented by any BPP.

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

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

$$\begin{array}{lll} X \xrightarrow{a} XA & XAB \xrightarrow{c} X & YA \xrightarrow{a} Y \\ X \xrightarrow{b} XB & X \xrightarrow{d} Y & YB \xrightarrow{b} Y \end{array}$$

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



Although in the sequential case, PDA constitutes a normal form for unrestricted rewrite transition systems, it is unlikely that this result holds in the parallel case. For example, we conjecture that there is no PPDA which represents an isomorphic transition system to that of the PN in Example 5.

2.1 Languages and Bisimilarity

Given a labelled transition system T with initial state α_0, 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 2.4 $L(\alpha) = \{ w \in \Sigma^* : \alpha \xrightarrow{w} \beta \text{ for some } \beta \in F \}$, 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*.

Definition 2.5 We define the *norm* of any state α of a labelled transition system, written $\mathrm{norm}(\alpha)$, 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 $\mathrm{norm}(\alpha) = \infty$ 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.

Note that, due to our assumption following Definition 2.2 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 BPA and BPP states.

Lemma 2.6 *Given any state $\alpha\beta$ of a type 2 rewrite transition systems (BPA or BPP),* $\mathrm{norm}(\alpha\beta) = \mathrm{norm}(\alpha) + \mathrm{norm}(\beta)$.

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

Definition 2.7 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 exhibitted 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 define by Park [43] and used to great effect by Milner [37, 38]. Its definition, in the presence of final states, is as follows.

Definition 2.8 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 $\mathcal{R}$.

Lemma 2.9 $\sim \; = \; \bigcup \left\{ \mathcal{R} \; : \; \mathcal{R} \textit{ is a bisimulation relation} \right\}$ *is the largest bisimulation relation, and is an equivalence relation.*

Bisimulation equivalence has an elegant characterisation in terms of certain two-player games [47]. 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 $\alpha \xrightarrow{a} \alpha'$ or $\beta \xrightarrow{a} \beta'$). 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 $\beta \xrightarrow{a} \beta'$ or $\alpha \xrightarrow{a} \alpha'$). 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 2.10 $\alpha \sim \beta$ *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. □

Also immediately evident then is the following lemma with its accompanying corollary relating bisimulation equivalence to language equivalence.

Lemma 2.11 *If $\alpha \sim \beta$ and $\alpha \xrightarrow{w} \alpha'$ with $w \in \Sigma^*$, then $\beta \xrightarrow{w} \beta'$ such that $\alpha' \sim \beta'$.*

Corollary 2.12 *If $\alpha \sim \beta$ then $\alpha \sim_L \beta$.*

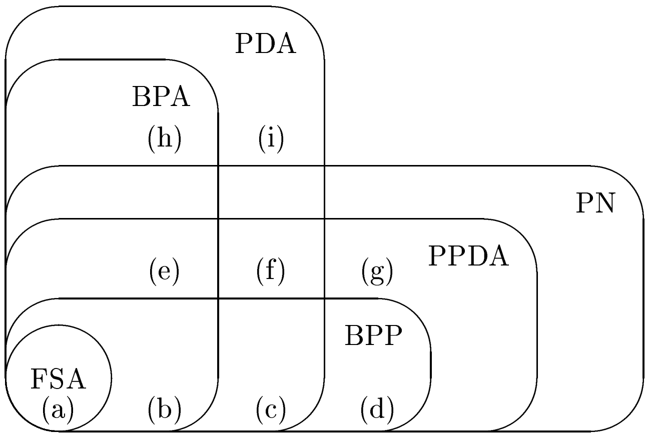
Apart from being the fundamental notion of equivalence for several process algebraic formalisms, bisimulation equivalence has several pleasing mathematical properties, not least of which being that it is decidable over classes of transition systems for which all other common equivalences, including language equivalence, remain undecidable. Furthermore as given by the following lemma, language equivalence and bisimilarity coincide over the class of normed deterministic processes.

Lemma 2.13 *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 2.12, $\sim_L$ and $\sim$ coincide.*

Hence it is sensible to concentrate on the more mathematically tractable bisimulation equivalence when investigating decidability results for language equivalence for deterministic language generators. In particular, by studying bisimulation equivalence we can rediscover old 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 techniques which can be exploited in the study of bisimulation equivalence will prove useful in tackling other language theoretic problems, notably the problem of deterministic push-down automata.

2.2 Expressivity Results

Our Chomsky 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 indicated in the classification. (We ignore the question of separating PN and PPDA, due to the uncertainty behind the existence of this gap; see the conjecture at the end of Section 2.) 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 [7].

(a) Example 0 provides a simple normed deterministic FSA.

(b) The rewrite system with the two rules $A \xrightarrow{a} AA$ and $A \xrightarrow{b} \varepsilon$ gives rise to the same rewrite transition system regardless of whether the rewrite system is sequential or parallel. Hence this is an example of a normed deterministic rewrite transition system which is both a BPA and a BPP but not an FSA.

(c) Example 2 provides 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 greater than 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 norm(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 $k = \text{norm}(\alpha)$.

(d) The following BPP with initial state X

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

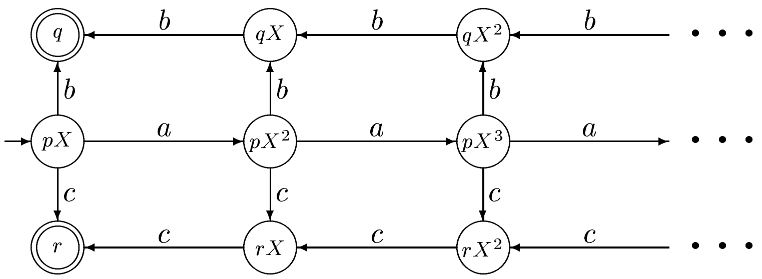
is not language equivalent to any PDA, as its language is easily confirmed not to be context free. (The words in this language from $a^*c^*b^*d^*e$ are exactly those of the form $a^kc^nb^kd^ne$, which is clearly not a context-free language.)

(e) Example 1 provides a transition system which can be described by both a BPA (Example 1) and a PPDA (Example 4). However, due to a pumping lemma for BPP given by Christensen [11] it is not language equivalent to any BPP.

(f) The following PDA with initial state pX

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

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



However, this transition system cannot be language equivalent to any BPP, due again to the pumping lemma for BPP [11], nor bisimilar to any BPA, due to a similar argument as for (c).

(g) The following PPDA with initial state pX

$$\begin{array}{llll} pX \xrightarrow{a} pA & pA \xrightarrow{a} pAA & qA \xrightarrow{b} qB & rA \xrightarrow{c} r \\ & pA \xrightarrow{b} qB & qB \xrightarrow{c} r & rB \xrightarrow{c} r \end{array}$$

generates the language $\{\, a^n b^k c^n \;:\; 0 < k \leq n \,\}$, and hence cannot be language equivalent to any BPP, due again to the pumping lemma for BPP [11], nor any PDA, as this language is not context-free.

(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 [44].

(i) The following PDA with initial state pX

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

like (h) above is not language equivalent to any PN, and like (c) above is not bisimilar to any BPA.

3 Decidability Results for Type 2 Rewrite Systems

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

1. Bisimilarity is decidable for BPA [14] and BPP [12, 13].
2. It is decidable in polynomial time for normed BPA [23, 24] and normed BPP [25].

These results contrast with those regarding the undecidability of language equivalence for both BPA and BPP. The negative result for BPA [3] follows from the fact that BPA effectively defines the class of context-free languages; and that for BPP [21] follows from a modification by Hirshfeld of a technique of Jančar which is described in Section 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 [16] between bisimilarity and language equivalence [31, 18, 28].

Baeten, Bergstra and Klop [1, 2] presented the first such decidability result, that bisimilarity for 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 [8], Hüttel and Stirling [29], and Groote [17]. Huynh and Tian [30] 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 [23, 24] 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 [26]. Finally, Christensen, Hüttel and Stirling [14, 15] demonstrated the general problem to be decidable, whilst Burkart, Caucal and Steffen [6] demonstrated an elementary decision procedure. For the parallel case, Christensen, Hirshfeld and Moller [12, 13] demonstrated the general decidability result for BPP, whilst Hirshfeld, Jerrum and Moller [25] presented a polynomial algorithm for the normed case.

One direction for decidability results, determining non-bisimilarity, is automatic using the following finite characterisation of bisimulation equivalence over *image-finite* transition systems, those for which there are only a finite number of transitions with a given label from any given reachable state.

Definition 3.1 The *stratified bisimulation relations* [38] $\sim_n$ 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 $\alpha' \sim_k \beta'$;

– $\alpha \in F$ iff $\beta \in F$.

Lemma 3.2 *If α and β are image-finite, then $\alpha \sim \beta$ iff $\alpha \sim_n \beta$ for all $n \geq 0$.*

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 our decidability results would follow from demonstrating the semi-decidability of bisimilarity. For this, we first note the following congruence results which hold for both BPA and BPP.

Fact 3.3 *$\sim$ is a congruence (with respect to catenation) over both BPA and BPP; that is, if $\alpha \sim \beta$ and $\alpha' \sim \beta'$ then $\alpha\alpha' \sim \beta\beta'$.*

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

Caucal [8] introduced the technique of using bisimulation bases for providing the decidability of normed BPA, by noting that the relation $\overset{\mathcal{R}}{\equiv}$ is a bisimulation whenever $\mathcal{R}$ is a bisimulation base; hence two states are bisimilar iff they are related by some bisimulation base. More than this we have the following finite base result for both BPA and BPP.

Fact 3.5 *$\alpha \sim \beta$ iff $(\alpha, \beta) \in \mathcal{R}$ for some finite bisimulation base $\mathcal{R}$, and it is semidecidable if a given finite relation $\mathcal{R}$ is a bisimulation base. Hence $\sim$ is semi-decidable.*

The statement of the above fact hides the complexity of its proof. The result for BPA is provided by Christensen, Hüttel and Stirling [15] and relies on a weak cancellation property, that $\alpha \sim \beta$ whenever $\alpha\gamma \sim \beta\gamma$ for infinitely many non-bisimilar γ. The result for BPP is most elegantly provided by Hirshfeld [22], though this is a revised proof using Dickson's Lemma [20] of a result of Redie [45] that every congruence on a finitely generated commutative semigroup (such as bisimilarity over BPP) is finitely generated. We note finally the desired result which follows from these finite base results.

Theorem 3.6 $\sim$ *is decidable for both BPA and BPP.*

When we turn our attention towards normed transition systems, we discover a useful unique decomposition result such as those considered by Milner and Moller [39, 40] for finite processes. Given a normed type 2 rewrite transition system, we say that a variable X is *prime* (with respect to bisimilarity) iff $\alpha = \varepsilon$ whenever $X \sim Y\alpha$.

Fact 3.7 *Every state of a normed type 2 rewrite transition system has a unique (up to bisimilarity) prime decomposition.*

Notice that this unique decomposition result immediately gives the finite base result from Fact 3.5 as a corollary. We merely take the finite relation $\mathcal{R}$ to relate each variable to its unique decomposition.

An obvious algorithm for deciding equivalence then is to compute the prime decompositions of the two states being compared and compare these. To compute the prime decompositions, we need first compute which of the variables are prime. This can be done iteratively, by assuming first that only the variables with norm 1 are prime (which they must be, due to Lemma 2.6), and then introducing primes as they are discovered (by deducing that they cannot be expressed in terms of the existing decompositions). By the use of careful bookkeeping this can be accomplished in both cases in polynomial time, thus giving our desired results.

Theorem 3.8 $\sim$ *is polynomial time decidable for both normed BPA and normed BPP.*

Hence the results for BPA and BPP are identical: bisimilarity is decidable in general and decidable in polynomial time for the normed cases, and these results follow from similar results regarding congruence and decomposability. However, this disguises the diversity in the actual details; interestingly, different techniques seem necessary for the two cases. In particular, care must be taken in the sequential case BPA as prime decompositions can be exponentially long.

An interesting corollary of the result for BPA is that language equivalence between simple (ie, deterministic) grammars is decidable in polynomial time. This result follows immediately from Lemma 2.13, and improves on the original doubly exponential algorithm of Korenjak and Hopcroft [34] as well as the singly exponential algorithm of Caucal [10].

As a final point, we note that the basic techniques which the above results rely upon, namely the congruence and decomposition properties, fail immediately for more general rewrite transition systems. As a simple example, if we take the following rewrite rules

$$A \xrightarrow{a} AA \qquad\qquad B \xrightarrow{a} A \qquad\qquad AA \xrightarrow{b} \varepsilon$$

then clearly $A \sim BA$. Hence congruence fails as $AA \not\sim BAA$, and cancellation (and hence any reasonable decomposition result) fails as $\varepsilon \not\sim B$. In this case, even Lemma 2.6

regarding the additivity of the norm fails, as norm$(A) = 2$ while norm$(AA) = 1$. Hence a more delicate analysis is necessary for attacking more general rewrite systems, such as for PDA. However, this can still be possible, as demonstrated by Stirling [48].

4 Undecidability Results for PPDA

In this section we demonstrate Jančar's technique [32] for demonstrating undecidability results for bisimilarity through mimicking Minsky machines in a weak fashion but faithfully enough to capture the halting problem. Jančar first employed his ideas to demonstrate the undecidability of bisimulation equivalence over the class of Petri nets. Hirshfeld [21] modified the argument to demonstrate the undecidability of trace equivalence over BPP. Jančar and Moller [33] 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) PPDA. Jančar's technique applies ideally in this case, as PPDA offer precisely the ingredients present in Petri nets which are needed to mimic Minsky machines.

Minsky Machines [41] are simple straight-line programs which make use of only two counters. Formally, a *Minsky machine* is a sequence of labelled instructions

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

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

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

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 $\mathtt{X}_\ell$ $(0 \le \ell < n)$, the machine executes instruction comm$_\ell$, modifying the contents of the counters and transferring the control to the appropriate label as directed by the instruction. The machine halts if and when the control reaches the `halt` instruction at label $\mathtt{X}_n$. 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 PPDA.

- The input alphabet is $\Sigma = \{ i, d, z, \omega \}$.
- The control states are $Q = \{ p_0, p_1, \ldots, p_{n-1}, p_n, q_0, q_1, \ldots, q_{n-1}, q_n \}$.
- The stack alphabet is $\Gamma = \{ Z, 0, 1 \}$.
- For each machine instruction

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

  we have the PPDA rules

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

- For each machine instruction

  $$\begin{array}{l} \mathtt{X}_\ell \ : \ \mathtt{if}\ \mathtt{c}_b = \mathtt{0}\ \mathtt{then}\ \mathtt{goto}\ \mathtt{X}_j \\ \qquad \mathtt{else}\ \mathtt{c}_b := \mathtt{c}_b\mathtt{-1;}\ \ \mathtt{goto}\ \mathtt{X}_k \end{array}$$

  we have the PPDA rules

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

- We have the one final PPDA rule

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

The two states $p_0 Z$ and $q_0 Z$ of this PPDA 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 PPDA 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 PPDA has only one transition available from the control state p_ℓ (q_ℓ), which is labelled by i (for 'increment'), resulting in the PPDA 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 PPDA has only one transition available from the control state p_ℓ (q_ℓ), which is labelled z (for 'zero'), again resulting in the PPDA 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 PPDA 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 PPDA state reflecting the state of the machine upon executing the decrement command.

- In this last instance, the PPDA 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 PPDA) 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 [35]). In this case, the PPDA 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 PPDA mimicking M.

Fact 4.1 *$p_0 Z \sim q_0 Z$ 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 PPDA states. Player II's only option in response would be to do the same with the other PPDA 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 [32] shows that bisimulation equivalence is decidable between two Petri nets when one of them is deterministic up to bisimilarity.

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

$$\begin{array}{lll} p_\ell X \xrightarrow{n} p_\ell & \qquad & q_\ell X \xrightarrow{n} q_\ell \\ p_\ell X \xrightarrow{n} q_\ell & \qquad & q_\ell X \xrightarrow{n} p_\ell \end{array}$$

These moves allow the PPDA 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 4.2 *Bisimilarity is undecidable over the class of normed PPDA.*

This result contrasts interestingly with that of Stirling [48] regarding the decidability of bisimilarity over the class of normed PDA.

References

[1] J.C.M. Baeten, J.A. Bergstra and J.W. Klop (1987). Decidability of bisimulation equivalence for processes generating context-free languages. Proceedings of PARLE'87, *Lecture Notes in Computer Science* **259**:94–113.

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

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

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

[5] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill and L.J. Hwang (1990). Symbolic model checking 10^{20} states and beyond. Proceedings of LICS'90:428–439.

[6] O. Burkart, D. Caucal and B. Steffen (1995). An elementary decision procedure for arbitrary context-free processes. Proceedings of MFCS'95. *Lecture Notes in Computer Science* **969**:423–433.

[7] O. Burkart, D. Caucal and B. Steffen (1996). Bisimulation collapse and the process taxonomy. This volume.

[8] D. Caucal (1990). Graphes canoniques des graphes algébriques. *Informatique Théorique et Applications (RAIRO)* **24**(4):339–352.

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

[10] D. Caucal (1993). A fast algorithm to decide on the equivalence of stateless DPDA. *Informatique Théorique et Applications (RAIRO)* **27**(1):23–48.

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

[12] S. Christensen, Y. Hirshfeld and F. Moller (1993). Decomposability, decidability and axiomatisability for bisimulation equivalence on basic parallel processes. Proceedings of LICS'93:386–396.

[13] S. Christensen, Y. Hirshfeld and F. Moller (1993). Bisimulation equivalence is decidable for basic parallel processes. Proceedings of CONCUR'93, *Lecture Notes in Computer Science* **715**:143–157.

[14] S. Christensen, H. Hüttel and C. Stirling (1992). Bisimulation equivalence is decidable for all context-free processes. Proceedings of CONCUR'92, *Lecture Notes in Computer Science* **630**:138–147.

[15] S. Christensen, H. Hüttel and C. Stirling (1995). Bisimulation equivalence is decidable for all context-free processes. *Information and Computation* **121**(2):143–148.

[16] R.J. van Glabbeek (1990). The linear time-branching time spectrum. Proceedings of CONCUR'90, *Lecture Notes in Computer Science* **458**:278–297.

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

[18] J.F. Groote and H. Hüttel (1994). Undecidable equivalences for basic process algebra. *Information and Computation* **115**(2):353–371.

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

[20] L.E. Dickson (1913). Finiteness of the odd perfect and primitive abundant numbers with distinct factors. *American Journal of Mathematics* **35**:413–422.

[21] Y. Hirshfeld (1993). Petri Nets and the Equivalence Problem. Proceedings of CSL'93, *Lecture Notes in Computer Science* **832**:165–174.

[22] Y. Hirshfeld (1994). Congruences in commutative semigroups. Research report ECS-LFCS-94-291, Department of Computer Science, University of Edinburgh.

[23] Y. Hirshfeld, M. Jerrum and F. Moller (1994). A polynomial-time algorithm for deciding equivalence of normed context-free processes. Proceedings of FOCS'94:623–631.

[24] Y. Hirshfeld, M. Jerrum and F. Moller (1996). A polynomial algorithm for deciding bisimilarity of normed context-free processes. *Theoretical Computer Science* **158**:143–159.

[25] Y. Hirshfeld, M. Jerrum and F. Moller (1996). A polynomial algorithm for deciding bisimulation equivalence of normed basic parallel processes. To appear in *Mathematical Structures in Computer Science*.

[26] Y. Hirshfeld and F. Moller (1994). A fast algorithm for deciding bisimilarity of normed context-free processes. Proceedings of CONCUR'94, *Lecture Notes in Computer Science* **836**:48–63.

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

[28] H. Hüttel (1993). Undecidable equivalences for basic parallel processes. Proceedings of FSTTCS'93, *Lecture Notes in Computer Science*.

[29] H. Hüttel and C. Stirling (1991). Actions speak louder than words: proving bisimilarity for context-free processes. Proceedings of LICS'91:376–386.

[30] D.T. Huynh and L. Tian (1994). Deciding bisimilarity of normed context-free processes is in Σ_2^{P}. *Theoretical Computer Science* **123**:183–197.

[31] D.T. Huynh and L. Tian (1995). On deciding readiness and failure equivalences for processes. *Information and Computation* **117**(2):193–205.

[32] P. Jančar (1993). Decidability questions for bisimilarity of Petri nets and some related problems. Proceedings of STACS'94, *Lecture Notes in Computer Science* **775**:581–592.

[33] P. Jančar and F. Moller (1995). Checking regular properties of Petri nets. Proceedings of CONCUR'95, *Lecture Notes in Computer Science* **962**:348–362.

[34] A. Korenjak and J. Hopcroft (1966). Simple deterministic languages. Proceedings of 7th IEEE Switching and Automata Theory conference:36–46.

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

[36] R. Milner (1984). A complete inference system for a class of regular behaviours. *Journal of Computer and System Science* **28**:439–466.

[37] R. Milner (1980). **A Calculus of Communicating Systems**. *Lecture Notes in Computer Science* **92**.

[38] R. Milner (1989). **Communication and Concurrency**. Prentice-Hall.

[39] R. Milner and F. Moller (1990). Unique decomposition of processes. *Bulletin of the European Association for Theoretical Computer Science* **41**:226–232.

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

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

[42] E.F. Moore (1956). Gedanken experiments on sequential machines. In *Automata Studies*:129–153.

[43] D.M.R. Park (1981). Concurrency and automata on infinite sequences. *Lecture Notes in Computer Science* **104**:168–183.

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

[45] L. Redei (1965). **The theory of finitely generated commutative semigroups.** Oxford University Press.

[46] A. Salomaa (1966). Two complete axiom systems for the algebra of regular events. *Journal of the ACM* **13**(1):158–169.

[47] C. Stirling (1995). Local model checking games. Proceedings of CONCUR'95, *Lecture Notes in Computer Science* **962**:1–11.

[48] C. Stirling (1996). Decidability of bisimulation equivalence for normed pushdown processes. This volume.

[49] D. Taubner (1989). **Finite Representations of CCS and TCSP Programs by Automata and Petri Nets**. *Lecture Notes in Computer Science* **369**.