Theoretical Computer Science 256 (2001) 3–21
Theoretical Computer Science
www.elsevier.com/locate/tcs

Pushdown automata, multiset automata, and Petri nets

Yoram Hirshfeld[a], Faron Moller[b,*]

[a]*School of Mathematical Sciences, Tel Aviv University, Ramat-Aviv 69978, Israel*
[b]*Computing Science Department, Uppsala University, P.O. Box 311, S-751 05 Uppsala, Sweden*

Abstract

In this paper, we consider various classes of (infinite-state) automata generated by simple rewrite transition systems. These classes are defined by two natural hierarchies, one given by interpreting concatenation of symbols in the rewrite system as sequential composition, and the other by interpreting concatenation as parallel composition. In this way, we provide natural definitions for commutative (parallel) context-free automata, multiset (parallel, or random access, pushdown) automata, and Petri nets. We provide example automata which demonstrate the strictness of this hierarchy. In particular, we provide a proof of an earlier conjecture by Moller: that multiset automata form a proper subset of Petri nets. This result contrasts with the result of Caucal for the analogous question in the sequential case where the hierarchy collapses.

Keywords: Automata; Bisimulation; Concurrency; Petri nets; Rewrite systems

1. Introduction

The behaviour of a system is modelled semantically in a variety of ways. It may be defined for example by the infinite traces or executions which it may perform, or by the entirety of the properties which it satisfies in some particular temporal logic, or as a particular algebraic model of some equational specification. In any case, a fundamental unifying view is to interpret a system as a *labelled transition system*, an edge-labelled directed graph whose nodes represent the states in which the 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.

* Corresponding author.
E-mail address: fm@csd.uu.se (F. Moller).

PII: S0304-3975(00)00099-2

In this paper, we shall consider various classes of such graphs as generated by two related hierarchies defined by simple rewrite transition systems. These two hierarchies differ only in the interpretation of the concatenation of symbols: in the first hierarchy concatenation is interpretted in the usual fashion as sequential composition, whereas in the second hierarchy concatenation is interpretted as parallel composition. Within these hierarchies, we find natural classes of automata such as the finite-state automata, context-free automata, pushdown automata, and Petri net automata; as well, we are provided with natural definitions for commutative (parallel) context-free automata and for multiset automata.

When considering the usual notion of language equivalence, this hierarchy collapses in certain places. For example, the classes of context-free automata and pushdown automata coincide with respect to the languages which they define: they both give rise to the class of context-free languages. However, while the context-free automata are but a restricted form of pushdown automata, there exist pushdown automata which are not isomorphic to any context-free automata. This result is originally due to Caucal and Montfort [8, 11], who demonstrated that the class of normed context-free automata is closed under bisimulation collapse (the identification of bisimilar states) while the class of normed pushdown automata is not. This final result exploits a characterisation of pushdown automata described by Muller and Schupp [44].

In this paper, we explore such questions within our double hierarchy. As part of this exploration, we provide a simple direct demonstration of the above result concerning the existence of the gap between context-free automata and pushdown automata. We give similarly simple demonstrations of the existence of gaps at various other borders. In particular, we settle positively a conjecture from Moller [42] that the class of multiset automata is strictly contained within the class of Petri nets. This result contrasts interestingly with the result of Caucal [9] (which we briefly demonstrate in this paper) that we have a collapse at the analogous point in the sequential hierarchy.

2. Rewrite transition systems

The starting point for our study will be automata, or labelled transition systems, as defined as follows.

Definition 1. A *labelled transition system* is a tuple $\langle S, \Sigma, \rightarrow, \alpha_0, F\rangle$ where

- S is a set of *states*.
- Σ is a finite set of *labels*.
- $\rightarrow \subseteq S \times \Sigma \times S$ is a *transition relation*, written $\alpha \xrightarrow{a} \beta$ for $\langle \alpha, a, \beta\rangle \in \rightarrow$.
- $\alpha_0 \in S$ 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 automaton (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 (i.e., deadlock) which is represented by all non-final terminal states. We could remove this restriction, but only at the expense of Theorem 6 below which characterises a wide class of labelled transition systems as pushdown automata which accept on empty stack. (An alternative approach could be taken to recover Theorem 6 based on PDA which accept by final state.)

In this overview, we follow the example set by Caucal [9] as extended in [42], and consider the families of labelled transition systems defined by various rewrite systems.

Definition 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 concatenation, which is thus interpretted as parallel, rather than sequential, composition.

We shall freely extend the transition relation $\rightarrow$ homomorphically to finite sequences of actions $w \in \Sigma^*$ so as to write $\alpha \xrightarrow{\varepsilon} \alpha$ and $\alpha \xrightarrow{aw} \beta$ whenever $\alpha \xrightarrow{a} \gamma \xrightarrow{w} \beta$ for some state $\gamma \in V^*$. Also, we shall 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 pushdown automata.

A natural hierarchy of families of transition systems can be defined by restricting the forms of the rewrite systems. This hierarchy is based loosely on the Chomsky hierarchy. (In this respect, 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

Table 1

	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

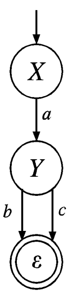
important classes of transition systems which have been defined and studied independent of their appearance as particular rewrite systems. This classification is presented in Table 1.

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 classes. In drawing labelled transition systems, initial states will be pointed to by a short arrow, and states will be presented either as single circles (for non-final states) or as double circles (for final states).

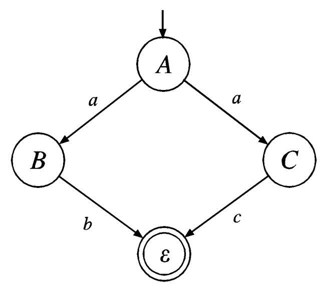
FSA represents the class of *finite-state automata*. Clearly, if the rules are restricted to be of the form $A \xrightarrow{a} B$ or $A \xrightarrow{a} \varepsilon$ with $A, B \in V$, then the reachable states of both the sequential and parallel transition systems will be a subset of the finite set of variables V. (We assume here that the initial state itself is a member of V, though this is clearly not necessary to demonstrate finiteness.)

Example 3. 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.

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



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

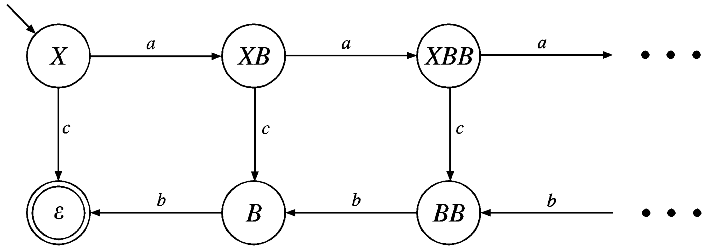


As language recognisers in the usual sense, these automata both recognise the same regular language (set of strings): $\{ab, ac\}$. However, they are substantially different automata.

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 4. 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.

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

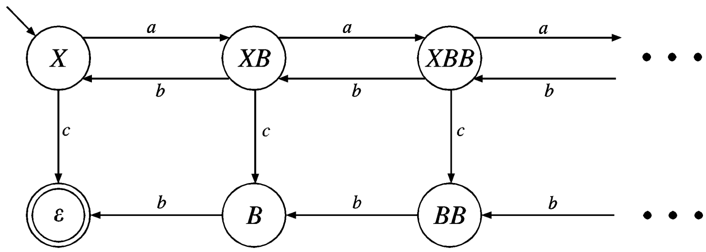


This automaton recognises the context-free language $\{a^n cb^n \colon n \geqslant 0\}$.

BPP represents the class of *Basic Parallel Processes* introduced by Christensen [12] 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 5. The type 2 rewrite system from Example 4 gives rise to the following BPP transition system with initial state X.

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



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

PDA represents the class of *pushdown 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 an arbitrary rewrite transition 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 $Q = \{ p_\alpha : \alpha \in V^*$ and $\mathrm{length}(\alpha) < n\}$ and $\Gamma = V \cup \{Z_\alpha : \alpha \in V^*$ and $\mathrm{length}(\alpha) \leqslant n\}$. Every final transition system 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 nonfinal transition system state $\alpha\beta\gamma$ with $\mathrm{length}(\alpha) < n$, $\mathrm{length}(\beta\gamma) > 0$ only if $\mathrm{length}(\alpha) = n-1$, and $\mathrm{length}(\beta) > 0$ only if $\mathrm{length}(\gamma) = n$, 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 transition system rewrite rule gives rise to appropriate PDA rules which mimic the transition system and respect this representation. Thus we arrive at the following result.

Theorem 6. *Every sequential labelled rewrite transition system can be represented (up to the labelling of states) by a PDA transition system.*

Example 7. The BPP transition system of Example 5 is given by the following sequential rewrite system:

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

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 4 that any PDA presentation of the BPP transition system of Example 5 must have at least two 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" pushdown automata. They are defined as above except that they have random access capability to the stack. Thus a MSA transition rule $pA \xrightarrow{a} q\beta$ with $p, q \in Q$, $A \in \Gamma$ and $\beta \in \Gamma^*$, says that while in control state p with the symbol A *anywhere* in the stack, you may read the input symbol a, move into control state q, and replace the stack element A with the sequence β.

Example 8. The BPA transition system of Example 4 is isomorphic to that given by the following MSA with initial state pX:

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

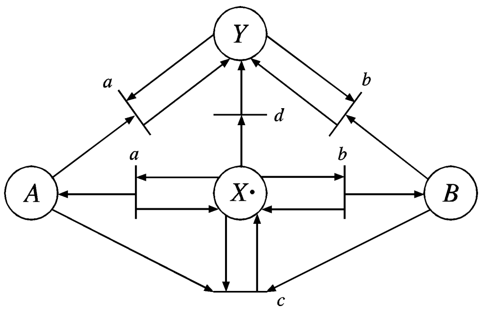
Note that when the stack alphabet has only one element, PDA and MSA trivially coincide. Also note that BPP coincides with the class of single-state MSA. However, we shall see in Section 4 that any MSA presentation of the BPA transition system of Example 4 must have at least 2 control states: this transition 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 9. 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 would be rendered as follows. (The weight on all 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, this result does not hold in the parallel case. For example, in [42] it was conjectured that there is no MSA which represents an isomorphic transition system to that of the PN in Example 9. In Section 4 we prove an even stronger negative result.

3. Languages and bisimilarity

Apart from isomorphism between transition systems, there are several other weaker notions of equivalence which are commonly studied. We shall be interested in two of these: language equivalence and bisimilarity. We have in fact already been describing the languages accepted by the automata in the examples of the previous section.

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 10. $L(\alpha) = \{w \in \Sigma^* : \alpha \xrightarrow{w} \beta$ 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: $L(\alpha) = L(\beta)$.

Thus, for example, the languages generated by FSA are precisely the (ε-free) regular languages; and the languages generated by both BPA and by PDA are the (ε-free) context-free languages.

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 11. We define the *norm* of any state α of a labelled transition system, written norm(α), 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 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 on the accessibility of all of 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 12. *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 13. 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 3 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 3 which generate the same language $\{ab, ac\}$ yet demonstrate different deadlocking capabilities due to the non-deterministic 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 defined by Park [46] and used to great effect by Milner [38, 39]. Its definition, in the presence of final states, is as follows.

Definition 14. A binary relation $\mathscr{R}$ on states of a transition system is a *bisimulation* iff whenever $\langle \alpha, \beta \rangle \in \mathscr{R}$ we have that
- if $\alpha \xrightarrow{a} \alpha'$ then $\beta \xrightarrow{a} \beta'$ for some β' with $\langle \alpha', \beta' \rangle \in \mathscr{R}$;
- if $\beta \xrightarrow{a} \beta'$ then $\alpha \xrightarrow{a} \alpha'$ for some α' with $\langle \alpha', \beta' \rangle \in \mathscr{R}$;
- $\alpha \in F$ iff $\beta \in F$.

α and β are *bisimulation equivalent* or *bisimilar*, written $\alpha \sim \beta$, iff $\langle \alpha, \beta \rangle \in \mathscr{R}$ for some bisimulation $\mathscr{R}$.

Lemma 15. $\sim\, = \bigcup \{\mathscr{R}$: *$\mathscr{R}$ is a bisimulation relation*$\}$ *is the largest bisimulation relation, and is an equivalence relation.*

Bisimulation equivalence has an elegant characterisation in terms of certain two-player games [50]. 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.

A *strategy* for one of the players for this game is a rule for determining which move that player should make at any point during the play of the game; and a *winning strategy* is one which guarantees a win regardless of the moves of the other player. Clearly, any bisimulation relation defines a winning strategy for player II for the 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. Furthermore, a winning strategy for player II for the 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. This observation leads immediately to the following result.

Fact 16. *$\alpha \sim \beta$ 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 $\langle \alpha, \beta \rangle$.*

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

Lemma 17. *If $\alpha \sim \beta$ and $\alpha \xrightarrow{w} \alpha'$ with $w \in \Sigma^*$, then $\beta \xrightarrow{w} \beta'$ such that $\alpha' \sim \beta'$.*

Corollary 18. *If $\alpha \sim \beta$ then $\alpha \sim_L \beta$.*

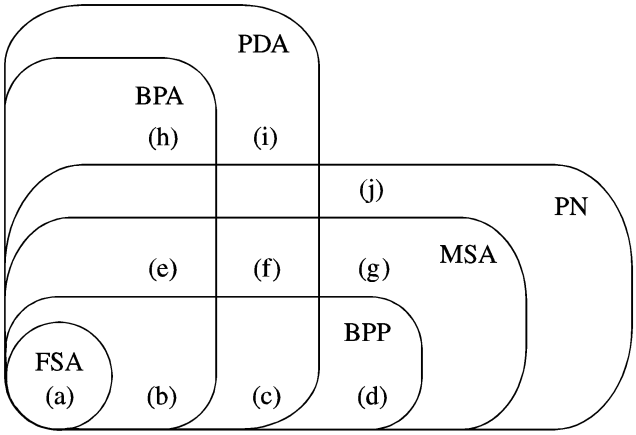
Apart from being the fundamental notion of equivalence for several process algebraic formalisms, bisimulation equivalence has several pleasing mathematical properties, not least of which being that it is decidable over classes of transition systems for which all other common equivalences, including language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

Lemma 19. *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* 18, *$\sim_L$ 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 finding a simple proof of the decidability of deterministic pushdown automata, for which a lengthy proof was recently demonstrated by Sénizergues [48].

4. Expressivity results

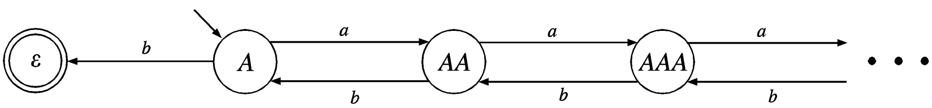
Our hierarchy from above gives us the following classification of processes.



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 in fact do more than this by giving examples of normed deterministic transition systems which separate all of these classes up to bisimulation (and sometimes even up to language) equivalence.

(a) The first transition system in Example 3 provides a normed deterministic FSA.

(b) 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 5 and 7 provide a transition system which can be described by both a BPP (Example 5) and a PDA (Example 7). 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 5 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 5 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

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

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 $a^k c^n b^k d^n e$, which is clearly not a context-free language.)

(e) Examples 4 and 8 provide a transition system which can be described by both a BPA (Example 4) and a MSA (Example 8). However, the context-free language which it generates, $\{a^n cb^n\colon\ n \geqslant 0\}$, cannot be generated by any BPP, so this transition system is not even language equivalent to any BPP. To see this, suppose that $L(X) = \{a^n cb^n\colon\ n \geqslant 0\}$ for some BPP state X. (As the process has unit norm, 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 $a^k cb^k$:

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

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

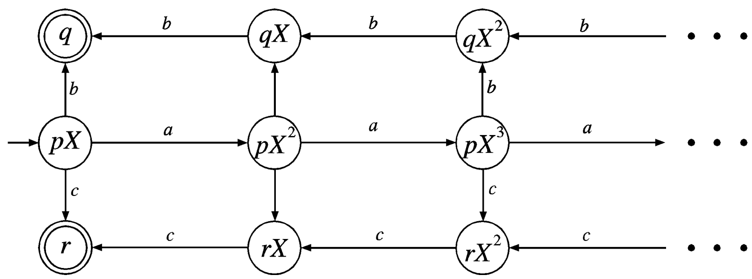
$$X \xrightarrow{a^k} Y\alpha \xrightarrow{b^i} Y \xrightarrow{c} \beta \xrightarrow{b^{k-i}} \varepsilon$$

from which we get our contradiction: $a^k b^i cb^{k-i} \in L(X)$ for some $i > 0$.

(f) The following PDA with initial state pX

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

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 $\{a^n b^k c^n\colon\ 0 < k \leqslant n\}$, 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 $\{wcw^{\mathrm{R}}\colon\ w \in \{a,b\}^*\}$ and hence is not language equivalent to any PN [47].

(i) The following PDA with initial state pX

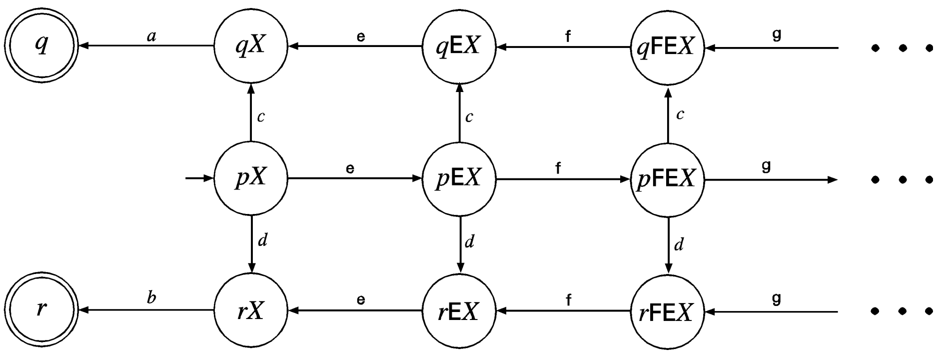
$$pX \xrightarrow{a} pAX \qquad pA \xrightarrow{a} pAA \qquad pB \xrightarrow{a} pAB \qquad qA \xrightarrow{a} q \qquad rA \xrightarrow{a} r$$

$$pX \xrightarrow{b} pBX \qquad pA \xrightarrow{b} pBA \qquad pB \xrightarrow{b} pBB \qquad qB \xrightarrow{b} q \qquad rB \xrightarrow{b} r$$

$$pX \xrightarrow{c} qX \qquad pA \xrightarrow{c} qA \qquad pB \xrightarrow{c} qB \qquad qX \xrightarrow{a} q \qquad rX \xrightarrow{b} r$$

$$pX \xrightarrow{d} rX \qquad pA \xrightarrow{d} rA \qquad pB \xrightarrow{d} rB$$

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



In this picture, $\mathsf{e}, \mathsf{f}, \mathsf{g}, \ldots \in \{a,b\}$ and $\mathsf{E}, \mathsf{F}, \mathsf{G}, \ldots \in \{A,B\}$ correspond in the obvious way. The language this PDA generates is $\{wcw^{\mathrm{R}}a,\ wcw^{\mathrm{R}}b\colon\ w \in \{a,b\}^*\}$ 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 9 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 $a^*b^*c^*d$ are exactly those of the form $a^nb^nc^nd$, which is clearly not a context-free language.)

More importantly, this Petri net cannot be bisimilar to any MSA. To see this, suppose that the net is bisimilar to the MSA state pA. (As the process has unit norm, the stack must consist of a single symbol A.) Consider performing an indefinite sequence of a-transitions from pA. By Dickson's Lemma [17], we must eventually pass through two states $q\alpha$ and $q\alpha\beta$ 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.

$$pA \xrightarrow{a^k} q\alpha \xrightarrow{a^k} q\alpha\beta \xrightarrow{a^k} q\alpha\beta^2 \xrightarrow{a^k} \cdots \ .$$

(We can assume that the period of the cycle is of the same length as the initial segment. If this is not 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 $q\alpha$, a second application of Dickson's Lemma gives us the following execution sequence.

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

(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 $s\sigma$ such that

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

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

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

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

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

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

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

Theorem 20. *There exist* (*normed and deterministic*) *labelled transition systems lying precisely in the gaps* (a)–(i) *in the figure above. In particular*, *there is a Petri net which is not even bisimilar to any MSA.*

This final result answers positively the conjecture made in [42], demonstrating that the class of Petri nets strictly contains the class of MSA. Intuitively, this is due to the ability of Petri nets – as demonstrated in the example – to maintain two independent unbounded counters which can cooperate in synchronisation transitions. This ability is absent from MSA. We have not demonstrated any gap between MSA and Petri nets within the class of PDA, as the above ability of Petri nets is also missing from PDA. Hence, we leave it as a conjecture that this gap collapses: that any Petri net which can be rendered as a PDA can also be rendered as a MSA.

5. Related work

The classes of transition systems represented within our double hierarchy have all occurred naturally in independent contexts. Indeed this is one of the beauties of the hierarchies: it gives a unified presentation of many classes that have been afforded a great deal of research. Some avenues of intense interest are as follows.

5.1. Further separability results

In this paper we have been interested in separating classes with respect to isomorphism between automata. We have however managed to demonstrate even stronger results, showing that classes could be separated up to bisimulation equivalence, and sometimes even up to language equivalence.

Of course, when we weaken the equivalence and equate more and more automata, this hierarchy will tend to collapse in expressivity. For example, BPA and PDA both express exactly the (ε-free) context-free languages, and hence the gap between BPA and PDA vanishes with respect to language equivalence. The question then is: which gaps are preserved with respect to language equivalence.

We have demonstrated in the previous section that most gaps are maintained apart from the BPA-PDA gap. For example, (h) shows that there are BPA languages which are not Petri net languages; (d) shows that there are BPP languages which are not BPA languages; and (g) shows that there are MSA languages which are not BPP languages. The only gap which remains to investigate is that between MSA and Petri nets. Recently, Hirshfeld [22] has settled this question by demonstrating that this gap vanishes with respect to language equivalence. He thus provides a new characterisation of Petri net languages in terms of MSA.

5.2. Equivalence checking

The first decidability result of relevance here regards language equivalence between finite-state automata [43]. The decidability of bisimulation is also readily established; but whereas the language equivalence problem is co-PSPACE-complete, bisimulation equivalence can be determined in time $O(k \lg n)$, where n and k are the total number of states and edges, respectively, of the two automata being compared [45, 36].

The first relevant result related to infinite-state automata is the undecidability of language equivalence between context-free automata BPA [3]. Groote and Hüttel [20] extend this undecidability result to all of the equivalences in van Glabbeek's catalogue of equivalences [18] except for bisimulation. Baeten et al. [1, 2] were the first to demonstrate that bisimulation is decidable for normed BPA. Their lengthy proof exploits the periodicity which exists in normed BPA transition systems, and several simpler proofs exploiting structural decomposition properties as introduced by Milner and Moller [40, 41] were soon recorded, notably by Caucal [8], Hüttel and Stirling [29], and Groote [19]. Huynh and Tian [30] demonstrate that this problem has a complexity of Σ_2^{P} by providing a non-deterministic algorithm which relies on an NP oracle; Hirshfeld et al. [23, 24] refine this result by providing a polynomial algorithm, thus showing the problem to be in P. As a corollary of this, we get a polynomial-time algorithm for deciding language equivalence of simple grammars, thus improving on the original doubly exponential algorithm of Korenjak and Hopcroft [37], and the singly exponential algorithm of Caucal [10]. A generally more efficient, though worst-case exponential, algorithm is presented by Hirshfeld and Moller [26]. Finally, Christensen et al. [15, 16] demonstrate the general problem to be decidable, whilst Burkart et al. [5] provide an elementary decision procedure.

For the case of commutative context-free automata BPP, we get similar results. Hirshfeld [21] demonstrates the undecidability of language equivalence, and Hüttel [28] extends this undecidability result to all of van Glabbeek's equivalences except bisimilarity. Christensen et al. [13, 14] demonstrate the decidability of bisimilarity, first for the normed case and then in the general case; and Hirshfeld et al. [25] provide a polynomial-time algorithm for the normed case.

For PDA, we note the recent positive solution of Sénizergues [49] to the long-standing question as to the decidability of language equivalence for deterministic PDA. (Note that this case includes the possibility of ε-transitions, which we have ignored in the present study.) A further recent result is the proof of Stirling [51] of the decidability of bisimilarity over normed PDA. The former proof is enormously long (exceeding 70pp. in its full, as yet unpublished form [49]); it would be worthwhile looking for an extension of the latter proof to provide a simpler demonstration of the classical problem, exploiting the coincidence of language and bisimulation equivalences over normed and deterministic automata.

Finally, for MSA and Peti nets, the results are more negative. Jančar [31, 32] demonstrates the undecidability of bisimilarity for Petri nets, and this result is refined in [42] to apply to the more restricted class MSA.

5.3. Minimizing automata and regularity checking

A further interesting question is that of regularity checking, that is, determining if an automaton is equivalent to some (unspecified) finite-state automaton. Often this question is addressed in conjunction with the question of minimizing automata, that is, collapsing equivalent states; the question then is if the collapsed automaton is finite, or if it even stays within the class of automata from which the original is taken.

Burkhart et al. [6] study the problem of bisimulation collapse for many of the classes of automata that we are considering. They determine that the classes are typically not closed under bisimulation collapse. However, one positive result which they obtain from their study is that regularity checking for BPA is decidable.

Valk and Vidal-Naquet [52] consider the regularity checking problem for Petri nets with respect to language (and trace) equivalence; and Esparza et al. [33–35] reconsider this problem particularly with respect to bisimulation equivalence, as well as the closely-related question of checking equivalence between a Petri net and a given finite-state automaton. The latter show that trace equivalence is decidable, even in the more general setting including ε-transitions, but that regularity checking with respect to trace equivalence is undecidable; this contrasts with the former's decidability result in the case that all labels on transitions (as appearing in the production rules) are unique. Finally, the latter demonstrate that the equivalence problem and regularity checking are both decidable with respect to bisimulation equivalence, but that both of these problems become undecidable when ε-transitions are permitted.

5.4. Model checking

The last topic we mention, but only briefly, is that of model checking: determining if a property expressed in some temporal logic holds of a given automaton. Typically, the logic in question is some subset of monadic second-order logic, such as the modal μ-calculus. As this paper has not addressed such questions, rather than list the myriad of results, we instead merely refer to the overview paper [7] which accompanied Esparza's invited lecture on the topic for Infinity'96.

References

[1] J.C.M. Baeten, J.A. Bergstra, J.W. Klop, Decidability of bisimulation equivalence for processes generating context-free languages, Proc. PARLE'87, Lecture Notes in Computer Science, vol. 259, Springer, Berlin, 1987, pp. 94–113.
[2] J.C.M. Baeten, J.A. Bergstra, J.W. Klop, Decidability of bisimulation equivalence for processes generating context-free languages, J. ACM 40 (1993) 653–682.
[3] Y. Bar-Hillel, M. Perles, E. Shamir, On formal properties of simple phrase structure grammars, Z. Phonetik, Sprachwissenschaft Kommunikationsforschung 14 (1961) 143–177.
[4] J.A. Bergstra, J.W. Klop, Algebra of communicating processes with abstraction, Theoret. Comput. Sci. 37 (1985) 77–121.
[5] O. Burkart, D. Caucal, B. Steffen, An elementary decision procedure for arbitrary context-free processes, Proc. MFCS'95, Lecture Notes in Computer Science, vol. 969, Springer, Berlin, 1995, pp. 423–433.
[6] O. Burkart, D. Caucal, B. Steffen, Bisimulation collapse and the process taxonomy, Proc. CONCUR'96, Lecture Notes in Computer Science, vol. 1119, Springer, Berlin, 1996, pp. 247–262.

[7] O. Burkart, Javier Esparza, More infinite results, Proc. Infinity'97, Electronic Notes in Theoretical Computer Science, vol. 5, 1997.
[8] D. Caucal, Graphes canoniques des graphes algébriques, Inform. Théor. Appl. (RAIRO) 24 (4) (1990) 339–352.
[9] D. Caucal, On the regular structure of prefix rewriting, J. Theoret. Comput. Sci. 106 (1992) 61–86.
[10] D. Caucal, A fast algorithm to decide on the equivalence of stateless DPDA, Inform. Théor. Appl. (RAIRO) 27 (1) (1993) 23–48.
[11] D. Caucal, R. Monfort, On the transition graphs of automata and grammars, Proc. Grath-theoretic Concepts in Computer Science, Lecture Notes in Computer Science, vol. 484, Springer, Berlin, 1990, pp. 311–337.
[12] S. Christensen, Decidability and decomposition in process algebras, Ph.D. Thesis ECS-LFCS-93-278, Department of Computer Science, University of Edinburgh, 1993.
[13] S. Christensen, Y. Hirshfeld, F. Moller, Decomposability, decidability and axiomatisability for bisimulation equivalence on basic parallel processes, Proc. LICS'93, 1993, pp. 386–396.
[14] S. Christensen, Y. Hirshfeld, F. Moller, Bisimulation equivalence is decidable for basic parallel processes, Proc. CONCUR'93, Lecture Notes in Computer Science, vol. 715, Springer, Berlin, 1993, pp. 143–157.
[15] S. Christensen, H. Hüttel, C. Stirling, Bisimulation equivalence is decidable for all context-free processes, Proc. CONCUR'92, Lecture Notes in Computer Science, vol. 630, Springer, Berlin, 1992, pp. 138–147.
[16] S. Christensen, H. Hüttel, C. Stirling, Bisimulation equivalence is decidable for all context-free processes, Inform. Comput. 121 (2) (1995) 143–148.
[17] L.E. Dickson, Finiteness of the odd perfect and primitive abundant numbers with distinct factors, Amer. J. Math. 35 (1913) 413–422.
[18] R.J. van Glabbeek, The linear time-branching time spectrum, Proc. CONCUR'90, Lecture Notes in Computer Science, vol. 458, Springer, Berlin, 1990, pp. 278–297.
[19] J.F. Groote, A short proof of the decidability of bisimulation for normed BPA processes, Inform. Process. Lett. 42 (1991) 167–171.
[20] J.F. Groote, H. Hüttel, Undecidable equivalences for basic process algebra, Inform. Comput. 115 (2) (1994) 353–371.
[21] Y. Hirshfeld, Petri nets and the equivalence problem, Proc. CSL'93, Lecture Notes in Computer Science, vol. 832, Springer, Berlin, 1993, pp. 165–174.
[22] Y. Hirshfeld, Unpublished, 1998.
[23] Y. Hirshfeld, M. Jerrum, F. Moller, A polynomial-time algorithm for deciding equivalence of normed context-free processes, Proc. FOCS'94 1994, pp. 623–631.
[24] Y. Hirshfeld, M. Jerrum, F. Moller, A polynomial algorithm for deciding bisimilarity of normed context-free processes, Theoret. Comput. Sci. 158 (1996) 143–159.
[25] Y. Hirshfeld, M. Jerrum, F. Moller, A polynomial algorithm for deciding bisimulation equivalence of normed basic parallel processes, Math. Struct. Comput. Sci. 6 (1996) 251–259.
[26] Y. Hirshfeld, F. Moller, A fast algorithm for deciding bisimilarity of normed context-free processes, Proc. CONCUR'94, Lecture Notes in Computer Science, vol. 836, Springer, Berlin, 1994, pp. 48–63.
[27] J.E. Hopcroft, J.D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, Reading, MA, 1979.
[28] H. Hüttel, Undecidable equivalences for basic parallel processes, Proc. TACS'94, Lecture Notes in Computer Science, vol. 789, Springer, Berlin, 1994, pp. 454–464.
[29] H. Hüttel, C. Stirling, Actions speak louder than words: proving bisimilarity for context-free processes, Proc. LICS'91 1991, pp. 376–386.
[30] D.T. Huynh, L. Tian, Deciding bisimilarity of normed context-free processes is in Σ_2^{P}, Theoret. Comput. Sci. 123 (1994) 183–197.
[31] P. Jančar, Decidability questions for bisimilarity of Petri nets and some related problems, Proc. STACS'94, Lecture Notes in Computer Science, vol. 775, Springer, Berlin, 1993, pp. 581–592.
[32] P. Jančar, Undecidability of bisimilarity for Petri nets and related problems, Theoret. Comput. Sci. 148 (1995) 281–301.
[33] P. Jančar, J. Esparza, Deciding finiteness of Petri nets up to bisimulation, Proc. ICALP'96, Lecture Notes in Computer Science, vol. 1099, Springer, Berlin, 1996, pp. 478–489.
[34] P. Jančar, J. Esparza, F. Moller, Petri nets and regular processes, J. Comput. System Sci. 59 (1999) 476–503.

[35] P. Jančar, F. Moller, Checking regular properties of Petri nets, Proc. CONCUR'95, Lecture Notes in Computer Science, vol. 962, 1995, pp. 348–362.
[36] P.C. Kanellakis, S.A. Smolka, CCS expressions, finite-state processes and three problems of equivalence, Inform. Comput. 86 (1990) 43–68.
[37] A. Korenjak, J. Hopcroft, Simple deterministic languages, Proc. 7th IEEE Switching and Automata Theory Conference, 1966, pp. 36–46.
[38] R. Milner, A Calculus of Communicating Systems, Lecture Notes in Computer Science, vol. 92, 1980.
[39] R. Milner, Communication and Concurrency, Prentice-Hall, Englewood Cliffs, NJ, 1989.
[40] R. Milner, F. Moller, Unique decomposition of processes, Bull. European Assoc. Theoret. Comput. Sci. 41 (1990) 226–232.
[41] R. Milner, F. Moller, Unique decomposition of processes, Theoret. Comput. Sci. 107 (357–363).
[42] F. Moller, Infinite results, Proc. CONCUR'96, Lecture Notes in Computer Science, vol. 1119, Springer, Berlin, 1996, pp. 195–216.
[43] E.F. Moore, Gedanken experiments on sequential machines, in: Automata Studies, C.E. Shannon, J. McCarthy (eds), Princeton University Press, 1956, pp. 129–153.
[44] D. Muller, P. Schupp, The theory of ends, pushdown automata and second order logic, Theoret. Comput. Sci. 37 (1985) 51–75.
[45] R. Paige, R.E. Tarjan, Three partition refinement algorithms, SIAM J. Comput. 16 (1987) 937–989.
[46] D.M.R. Park, Concurrency and automata on infinite sequences, Lecture Notes in Computer Science, vol. 104, Springer, Berlin, 1981, pp. 168–183.
[47] J.L. Peterson, Petri Net Theory and the Modelling of Systems, Prentice-Hall, Englewood Cliffs, NJ, 1981.
[48] G. Sénizergues, The equivalence problem for deterministic pushdown automata is decidable, Proc. ICALP'97, Lecture Notes in Computer Science, vol. 1256, Springer, Berlin, 1997, pp. 671–681.
[49] G. Sénizergues, L(A) = L(B)? Technical Report, LaBRI, Université Bordeaux I, Report Nr.1161-97, 1997.
[50] C. Stirling, Local model checking games, Proc. CONCUR'95, Lecture Notes in Computer Science, vol. 962, Springer, Berlin, 1995, pp. 1–11.
[51] C. Stirling, Decidability of bisimulation equivalence for normed pushdown processes, Proc. CONCUR'96, Lecture Notes in Computer Science, vol. 1119, Springer, Berlin, 1996, pp. 217–232.
[52] R. Valk, G. Vidal-Naquet, Petri nets and regular languages, J. Comput. System Sci. 23 (3) (1981) 299–325.