On the Computational Complexity of Bisimulation, Redux [1]

Faron Moller [a], Scott Smolka [b,*], Jiří Srba [c,2]

[a] *Dept of Computer Science, University of Wales Swansea, Singleton Park, Swansea SA2 8PP, Wales.*

[b] *Dept of Computer Science, SUNY at Stony Brook, Stony Brook, NY 11790-4400, USA.*

[c] *BRICS, Aalborg University, Fr. Bajersvej 7B, 9220 Aalborg, Denmark.*

Abstract

Paris Kanellakis and the second author (Smolka) were among the first to investigate the computational complexity of bisimulation, and the first and third authors (Moller and Srba) have long-established track records in the field. Smolka and Moller have also written a brief survey about the computational complexity of bisimulation [45]. The authors believe that the special issue of *Information and Computation* devoted to *PCK50: Principles of Computing and Knowledge: Paris C. Kanellakis Memorial Workshop* represents an ideal opportunity for an up-to-date look at the subject.

Key words: Automata, formal languages, bisimulation equivalence, complexity

1 Introduction

In his Turing Award lecture [17], Juris Hartmanis eloquently discusses, among other things, the fundamental rôle that *computational complexity* theory plays in computer science. He goes on, in the context of describing joint work with Phil Lewis and Richard Stearns, to highlight some of the results obtained on

* Corresponding author.
Email addresses: F.G.Moller@swansea.ac.uk (Faron Moller), sas@sunysb.edu (Scott Smolka), srba@brics.dk (Jiří Srba).

1 A preliminary version of this paper appeared in [46].
2 Supported by the Grant Agency of the Czech Republic, Grant No. 201/03/1161.

the computational complexity of problems in formal language theory; e.g., all context-free languages are contained in TIME[n^3] and SPACE[$\log^2 n$].

We argue here that the computational complexity of *generative devices* such as grammars or automata takes on a new and interesting light when such devices are interpreted as generating (concurrent) *processes* rather than formal languages, and the traditional notion of language equivalence is replaced by some notion of *semantic equivalence*. When employing grammars to generate processes, we assume them to be in Greibach Normal Form (GNF). In this way, a state of a process corresponds to a sequence of nonterminals; and the transitions leading from a state corresponding to a sequence starting with a nonterminal X are prescribed, in a one-to-one fashion, by the rules of the grammar corresponding to the nonterminal X. Formally, $X\beta \xrightarrow{a} \alpha\beta$ if $X \to a\alpha$ is a rule of the grammar. In concurrency theory such a transition is read as "process $X\beta$ performs the action a and evolves into the process $\alpha\beta$".

A wide range of semantic equivalences was classified by van Glabbeek [67,68] in his linear time/branching time spectrum (see Figure 1). The coarsest (least discriminating) equivalence in this hierarchy is *trace equivalence*, as defined by Hoare [23]. A (partial) trace of a process is a finite sequence of actions that can be performed by the process. Two processes are trace equivalent if their sets of traces are equal.

A variant of trace equivalence is called *completed trace equivalence*, which in the theory of formal languages and automata is known as *language equivalence* (provided that infinite computations are disregarded). A completed trace is maximal in the sense that it cannot be extended to a longer one. Two processes are completed trace equivalent if they are trace equivalent and moreover they have the same set of completed traces.

As we go further up in van Glabbeek's spectrum, the equivalences distinguish more and more branching features. The finest (most discriminating) equivalence is *bisimulation equivalence*, or *bisimilarity*. This is perhaps *the* equivalence that has attracted the most attention in concurrency theory, and is also the main focus of this paper. As originally introduced by Park [48] and Milner [39], bisimilarity appeared to play a prominent rôle due to many pleasant properties it possesses.

Bisimulation equivalence is the cornerstone of a number of theories of concurrent and distributed computing, most notably Robin Milner's Calculus of Communicating Systems (CCS) [41] and the π-calculus [43]. Milner received the 1991 Turing Award, and bisimulation figured prominently in his Turing Award lecture [42].

The idea underlying bisimulation equivalence had also drawn the attention of modal logicians already 30 years ago, in the guise of p-morphisms [51],

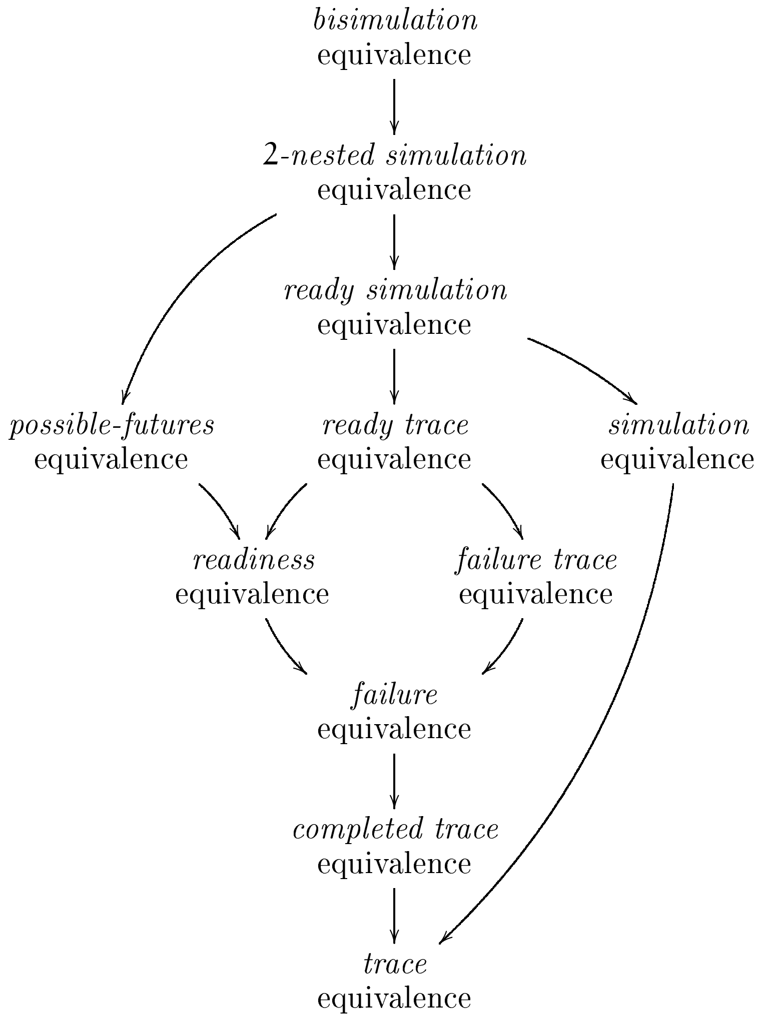


Fig. 1. Linear/branching time spectrum

and later zig-zag relations [66]. It is intimately related to the distinguishing powers of general branching-time temporal logics, in particular the modal mu-calculus. Set theorists have been attracted to bisimulation, as it forms the basis of Peter Aczel's Anti Foundation Axiom for non-well-founded set theory [7]. The functional programming community has also shown interest in bisimulation, as evidenced by Samson Abramsky's notion of *applicative bisimulation* for relating terms of the lazy lambda calculus [1].

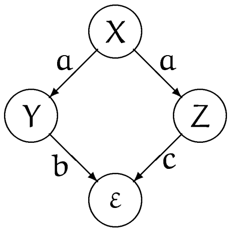
The essence of bisimilarity, quoting Hennessy and Milner [18], "is that the behaviour of a program is determined by how it communicates with an observer." Therefore, the notion of bisimilarity for different models is defined in terms of their *behaviours* and *observable behaviours*. For example for rooted labelled transition systems it seems natural to identify their behaviours with (possibly infinite) synchronization trees [39] into which they unfold, and to take sequences of actions as observations. The abstract definition of bisimilarity for arbitrary categories of models due to Joyal, Nielsen and Winskel [33] formalizes this idea. Given a category of models where objects are behaviours and morphisms indicate how one behaviour extends the other, and given a subcategory of observable behaviours, the abstract definition yields a notion of bisimilarity for all behaviours with respect to observable behaviours. For example, for rooted labelled transition systems, taking synchronization trees as their behaviours, and sequences of actions as the observable behaviours, we recover the standard notion of strong bisimilarity.

Another abstract definition of bisimilarity is that based on coalgebras. Transition systems of various kinds can be viewed as coalgebras for appropriate endofunctors. This approach gives rise to a definition of bisimulation as a span of coalgebras [65,50].

A remarkable property of bisimilarity is its computational feasibility. Bisimilarity is widely regarded as the "most decidable" behavioural equivalence, and this aspect will be demonstrated in the rest of this paper.

Finally, as we shall show, bisimulation has an elegant game-theoretic interpretation as promulgated by Stirling [59] and Thomas [64].

To motivate this study, consider the two regular expressions $\mathsf{ab} + \mathsf{ac}$ and $\mathsf{a}(\mathsf{b} + \mathsf{c})$ which are represented by the following nondeterministic finite-state automata (NFA) and corresponding regular grammars.

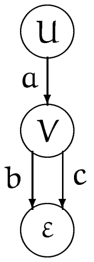


$\mathsf{X} \to \mathsf{aY}$

$\mathsf{X} \to \mathsf{aZ}$

$\mathsf{Y} \to \mathsf{b}$

$\mathsf{Z} \to \mathsf{c}$



$\mathsf{U} \to \mathsf{aV}$

$\mathsf{V} \to \mathsf{b}$

$\mathsf{V} \to \mathsf{c}$

These expressions, as well as their corresponding automata, are clearly language equivalent, as they both describe the language $\{\,\mathsf{ab},\ \mathsf{ac}\,\}$; as language generators, they are indistinguishable. However, viewed as process generators they may be distinguished. The first automaton in its initial state X may perform an a-transition and evolve into either state Y—from which only a b-transition is possible—or state Z—from which only a c-transition is possible; on the other hand, the second automaton in its initial state U performs the a-transition and evolves into state V from which both the b- and c-transitions are possible.

If we interpret these automata as representing the behaviours of processes, with the transitions being potential communications with the environment in which these processes reside, then it behooves us to consider them as behaviourally *in*equivalent. For example after the initial communication involving the a-transition, the second process would be in state V from which it is willing to participate in a communication involving a b-transition, whereas the first process may be in state Z from which it will refuse to participate in a communication involving a b-transition. In the terminology of concurrency theory, the first process may *deadlock* in an instance in which the second will not.

Milner [39] proposed bisimilarity to formally capture the notion of behavioural equivalence, and gave it, along with Park [48], a simple and elegant mathematical definition in terms of bisimulations. A *(strong) bisimulation* is a binary relation $\mathcal{R}$ on processes such that whenever $\mathcal{R}(\mathsf{P},\mathsf{Q})$: if P can perform an a-transition to become P' (for any a and P'), then Q can also perform an a-transition to become some Q' such that $\mathcal{R}(\mathsf{P}',\mathsf{Q}')$; and conversely, if Q can perform an a-transition to become Q', then P can also perform an a-transition to become some P' such that $\mathcal{R}(\mathsf{P}',\mathsf{Q}')$. Note the recursive nature of the definition. Now, two processes P and Q are *bisimilar* if there exists a bisimulation $\mathcal{R}$ such that $\mathcal{R}(\mathsf{P},\mathsf{Q})$. It is well-known that bisimulations are closed under union and that the largest bisimulation, under set inclusion, exists. In fact, this largest bisimulation, $\sim$, is an equivalence relation and taken to be bisimulation equivalence, or simply (strong) bisimilarity.

It is instructive to view bisimulation equivalence in terms of particular two-player games. A *game* is provided by a pair of processes (P,Q), with the players alternating moves as follows: Player I chooses a sequence of transitions of one of the processes, and in response, Player II must choose an identically labelled sequence of transitions of the other process. The game then continues starting from the resulting pair of processes. If Player II ever finds that she cannot respond to a move made by Player I, she loses the game. On the other hand, if Player I cannot perform any transition from either of the processes, the Player II wins. If the game is infinite, Player II is the winner. A moment's reflection then leads to the realization that Player II has a defending strategy exactly when the two processes are bisimilar: if there is a bisimulation relating the processes, then a defending strategy for Player II consists of merely matching transitions made by Player I which lead to a resulting pair which is also contained in the bisimulation relation. Conversely Player I has a winning strategy exactly when the two processes are not bisimilar.

As an example, the two processes X and U pictured above are not bisimilar. (There is no bisimulation relating X and U.) This nonbisimilarity is evidenced by the existence of an obvious winning strategy for Player I in the game defined by the pair (X,U). After one exchange of moves consisting of a single a-transition, the game must be in either the configuration (Y,V) or (Z,V). In the first instance, Player I may win by choosing the single c-transition from process V, while in the second instance she may win by choosing the single b-transition from process V. Thus, bisimilarity is a strictly more discriminating equivalence relation than language equivalence, and is intrinsically sensitive (unlike language equivalence) to the nondeterministic branching structure of processes.

For some applications the notion of bisimilarity is too strong because it reflects not only the *visible (observable) actions* but also the *internal (unobservable) actions*. This means that two processes have to exhibit bisimilar behaviours including, e.g., internal synchronization. It is, however, often not desirable to observe such internal events (e.g. implementation hiding in software-engineering). For this reason a special *silent action*, generally denoted by τ, is introduced with the intention that the action τ should be undetectable by an external observer.

Several semantic approaches can differ in the way they treat the unobservable action τ. One possibility is to disregard τ actions and agree that only the visible actions are observable. Citing Milner [41]: "...we merely require that each τ action is matched by *zero* or more τ actions ...". The notion of bisimilarity achieved this way is called *weak bisimilarity*.

In order to define weak bisimilarity one usually introduces the so-called *weak transition relation*. The idea is that a process P performs under the weak transition relation an action a and evolves into a process Q whenever it is possible to perform from P zero or more τ actions and then the action a, followed again by zero or more τ actions, finally resulting in Q. We also allow that P under the weak transition relation performs the τ action and evolves into P again.

Weak bisimilarity then corresponds to the bisimilarity notion defined above where instead of the basic transition relations we use the weak transition relations. In the same manner one can also generalize the bisimulation game described above.

Let us also mention that in [69] van Glabbeek and Weijland introduced a finer notion of behavioural equivalence than weak bisimilarity called *branching bisimilarity*. Their approach builds on the ideas of weak bisimilarity but it moreover distinguishes between processes that change their branching properties after the performance of individual τ-actions. This in particular means that if a τ-action is performed by one of the processes then the other process not only has to match this move by a sequence of τ actions but also all the intermediate states reached during this sequence have to be branching bisimilar to the first process.

For completeness let us mention that at least two other behavioural equivalences that abstract away from unobservable actions, called *eta bisimilarity* [3] and *delay bisimilarity* [40], have been proposed. They treat abstraction from unobservable actions in a slightly different way than branching bisimilarity and are positioned between weak and branching bisimilarity, mutually incomparable.

Since this paper deals only with strong and weak bisimilarity, we shall not provide further details about the other notions of bisimilarity. The interested reader is referred to [70].

We now have in place the three main ingredients of a formal language theory in a new setting: automata and grammars (processes), and equivalence (bisimilarity). The computational complexity of bisimulation in this formal-language framework, however, differs greatly from its classical counterpart, with a number of surprising twists and turns worth mentioning. We concentrate here on the inner layers of the Chomsky hierarchy, viz. *regular* and *context-free* processes, and note in passing that a language like CCS is easily shown to be Turing-powerful.

2 Regular Processes

In the case of regular processes, that is, those given by right-linear GNF grammars such as the two depicted above, the main complexity result is as follows. Let P,Q be regular processes whose underlying NFA have a total of n states and m transitions. Then, as was shown by Kanellakis and Smolka [34], whether or not P and Q are bisimilar can be decided in polynomial time, $O(nm)$ time to be exact. This algorithm was subsequently improved upon by Paige and Tarjan who devised one that runs in $O(m \log n)$ time [47]. This is in stark contrast to the equivalence problem for regular expressions, which was shown to be PSPACE-complete [24]. The class of regular processes is usually denoted by FS, to emphasize the intrinsically finite-state nature of these processes.

Moreover, bisimulation was originally defined by Milner as the limit of a sequence of successively finer equivalence relations, $\sim_k$, where $\sim_1$ is trace equivalence. In terms of our game-theoretic characterisation, two processes are related by $\sim_k$ exactly when Player II has a winning strategy if the game is redefined to declare her the winner after the exchange of k moves. So, for example, the above processes X and U are related by $\sim_1$ but not by $\sim_2$ as Player I has a strategy for guaranteeing a win within the exchange of two moves. Kanellakis and Smolka showed that, for each fixed k, deciding $\sim_k$ is PSPACE-complete, a complexity that disappears in the limit; i.e., upon reaching $\sim$.

As for weak bisimilarity on regular processes, one can first pre-compute the weak transition relation (which simply amounts to computing of the transitive closure) and construct new regular processes where transitions are replaced with weak transitions. On these new regular systems the algorithms for (strong) bisimilarity checking can be used. Hence the problem for weak bisimilarity can also be decided in polynomial time.

3 Context-Free Processes

The situation is even more dramatic in the context-free case, where the resulting processes are no longer regular. In the concurrency theory community, context-free processes are referred to as BPA (Basic Process Algebra) processes. In the classical setting, Bar-Hillel, Perles, and Shamir [6] showed that the equivalence problem for languages generated by nondeterministic context-free grammars is undecidable. In fact all the equivalences in van Glabbeek's spectrum (apart from bisimilarity) are undecidable for BPA [26,16].

Taking advantage of the periodic structure exhibited by bisimilar processes, Baeten, Bergstra, and Klop [2] were able to show that bisimilarity of *normed* BPA—those context-free processes in which the underlying GNF grammar contains no redundant nonterminals—is decidable. (Being normed means that there is a sequence of transitions leading from any state to the state ε. The norm of a state is defined as the length of the shortest such sequence.) In fact, Hirshfeld, Jerrum and Moller [21] showed that, in this case, bisimilarity can be decided in polynomial time. Restricting to simple (i.e., deterministic) normed grammars, where language equivalence and bisimilarity coincide, this gives that language equivalence is polynomially decidable, improving vastly on the doubly-exponential algorithm of Korenjak and Hopcroft [36].

For arbitrary (unnormed) BPA processes, Christensen, Hüttel, and Stirling [14] showed that bisimilarity is still decidable. However, the complexity in this general case is now known to be PSPACE-hard [56], yet no worse than doubly-exponential [10].

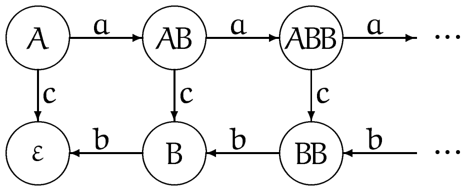
Decidability of weak bisimilarity checking for BPA is still an open problem. It is generally conjectured that the problem is decidable but so far only a partial positive result for a restricted subclass of totally normed BPA was achieved by Hirshfeld in [20]. However, the problem was very recently shown to be at least EXPTIME-hard by Mayr [38], even for normed BPA.

4 Commutative Context-Free Processes

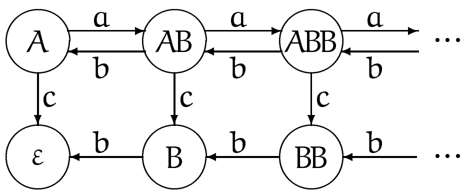
Of course, in studying concurrent processes one would like to consider processes composed not merely sequentially as with context-free processes, but concurrently as well. A simple form of concurrent composition can be modelled by considering commutative context-free processes; that is, where we now interpret concatenation of nonterminals modulo commutativity. In this way, any nonterminal in a sequence can be used to provide the next transition from the state associated with that sequence. In the concurrency theory community, the resulting process is referred to as BPP (Basic Parallel Process). For example, the grammar

$$A \to aAB \qquad A \to c \qquad B \to b$$

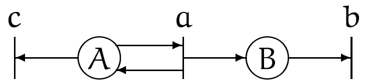
gives rise to the BPA (context-free) process



and to the BPP (commutative context-free) process



BPP processes correspond to communication-free Petri nets, in other words those (place/transition) Petri nets in which each transition has a unique input place. For example, the above BPP process corresponds to the following Petri net:



The results regarding deciding bisimulation in this case are similar to those for BPA processes. Hirshfeld [19] showed that once again language equivalence is undecidable (in fact none of the equivalences from van Glabbeek's spectrum below bisimilarity are decidable [25]), while Christensen, Hirshfeld and Moller [12] showed that bisimilarity is decidable in general, and Hirshfeld, Jerrum and Moller [22] showed that it is decidable in polynomial time for normed processes. The problem for unnormed BPP is known to be PSPACE-hard [55] and Jančar [29] has recently demonstrated that it can indeed be decided in polynomial space. PSPACE-completeness of the problem is hence established.

One noteworthy corollary from Jančar's paper is the resolution of an intriguing long-standing conjecture. In the case of BPA processes, if X is an unnormed variable it is easily confirmed that $X \sim X\alpha$ for any α; being unnormed, the process represented by the nonterminal X can never terminate, so the behaviour of $X\alpha$ will be the same as that of X. Also, if $XX \sim XXX$ then the variable X must be unnormed; this follows from the fact that the norm is additive, and bisimilar processes must have the same norm. Thus it is clear that the identity $X \sim XX$ follows immediately from $XX \sim XXX$. However, this is by no means obvious in the case of BPP processes. This conjecture was put forward more than a decade ago (a stronger version appears in [13]), and since then many clever researchers have failed at every attempt to prove this cancellation law; none of the standard bisimulation proof techniques could be applied to this question. Jančar finally provided a proof which is unexpectedly complicated for such a simple-looking conjecture.

Even though the weak bisimilarity problem for BPP is still open, Jančar conjectured that his new technique from [29] might be extended to prove decidability of this problem. His conjecture is also partially confirmed by positive decidability results of weak bisimilarity for several subclasses of BPP [20,62].

5 State-Extended Processes

A common extension to context-free and commutative context-free processes is provided by including a finite-state control unit. With such an extension, the grammar rules are no longer based solely on a nonterminal from the sequence representing the state, but are dictated as well by the finite-state control. State-extended BPA naturally correspond to pushdown automata (PDA), with the nonterminal sequence representing the stack, while state-extended BPP correspond to multiset automata (MSA), which are sometimes referred to as PPDA for parallel pushdown automata, and represent a subclass of Petri nets.

Sénizergues [52] and Stirling [60] both showed the decidability of bisimulation equivalence over state-extended BPA. Another noteworthy result is the result of Stirling [61] that bisimilarity is decidable over strict deterministic grammars. This result reinforces (and gives a shorter proof for) Sénizergues's solution [53] to the long-standing equivalence problem for deterministic pushdown automata. Recently Stirling proved that this problem is primitive recursive [63].

For the case of state-extended BPP, the result differs from the sequential case; here bisimilarity was proved undecidable [44] using Jančar's technique for the undecidability of bisimilarity of Petri nets [28].

Very recently the weak bisimilarity problem for PDA was shown to be undecidable [57], and it was proved that weak bisimilarity of Petri nets and MSA is significantly harder than strong bisimilarity. In fact, the weak bisimilarity problems are highly undecidable both for PDA and MSA [58] (Σ^1_1-complete in the analytical hierarchy), which contrasts to decidability of strong bisimilarity for PDA [52,60] and to Π^0_1-completeness (first level of the arithmetical hierarchy) of strong bisimilarity for Petri nets and MSA (see [27]).

6 Process Rewrite Systems

Based on the models of infinite-state systems introduced above, a successful effort to provide a common framework for their analysis was started by Moller in [44], and a slightly generalized and simplified version of this formalism was presented by Mayr [37] in the form of *Process Rewrite Systems* (PRS).

In the PRS formalism processes are identified with *process expressions* which consist of atomic *process constants* combined into larger process expressions by means of *sequential* and *parallel* operators. Formally the class of general process expressions $\mathcal{G}$ is defined by the following abstract syntax

$$\mathsf{E} \ ::= \ \varepsilon \ \mid \ \mathsf{X} \ \mid \ \mathsf{E.E} \ \mid \ \mathsf{E|E}$$

where 'ε' denotes the *empty process*, X ranges over a given set of process constants, and '.' and '|' are the operators of sequential and parallel composition, respectively. Moreover, we assume that '.' is associative, '|' is associative and commutative, and 'ε' is a unit for '.' and '|'.

A *process rewrite system* is a finite set Δ of rewrite rules of the form $\mathsf{E} \xrightarrow{a} \mathsf{E}'$ such that E and E' are from $\mathcal{G}$, and a is from a given set of actions. This finite set of rewrite rules generates an infinite-state process by means of the following inference rules (recall that '|' is commutative).

$$\frac{(\mathsf{E} \xrightarrow{a} \mathsf{E}') \in \Delta}{\mathsf{E} \xrightarrow{a} \mathsf{E}'} \qquad \frac{\mathsf{E} \xrightarrow{a} \mathsf{E}'}{\mathsf{E.F} \xrightarrow{a} \mathsf{E}'.\mathsf{F}} \qquad \frac{\mathsf{E} \xrightarrow{a} \mathsf{E}'}{\mathsf{E|F} \xrightarrow{a} \mathsf{E}'|\mathsf{F}}$$

The intuition behind the combinations of parallel and sequential operators on the left- and right-hand sides of the rewrite rules is as follows. A rewrite rule of the form $\mathsf{X} \xrightarrow{a} \mathsf{Y}$ can be interpreted as "process X performs the action a and becomes process Y." Similarly a rewrite rule of the form $\mathsf{X} \xrightarrow{a} \varepsilon$ means that "process X performs the action a and terminates." The interpretation of the rewrite rule $\mathsf{X} \xrightarrow{a} \mathsf{Y.Z}$ is "process X calls a procedure Y and then continues as process Z". When the sequential operator is present on the left-hand side of a rewrite rule as in $\mathsf{X.Y} \xrightarrow{a} \mathsf{Z}$, the intuition is that we enable value passing: X

represents a value returned by some previous computation and the behaviour of the process Y is affected by this value.

We now proceed to discuss the parallel operator '$|$'. A rewrite rule of the form $X \xrightarrow{a} Y|Z$ stands for "process X performs the action a and becomes a parallel composition of processes Y and Z." In other words, process X forks into Y and Z. Rewrite rules of the form $X|Y \xrightarrow{a} Z$ are interpreted as "processes X and Y synchronize by jointly executing the action a and becoming the process Z."

Finally, in the most general cases, we allow process expressions to contain a mixture of sequential and parallel operators on both sides of the rewrite rules, as in the rule

$$X.Y \xrightarrow{a} (U|V).Z.$$

This particular rule can be interpreted as follows: "process Y receives a return value X and performs the action a; after this, a parallel execution of U and V is initiated; finally, when both of these parallel components terminate, the computation continues with the execution of the process Z."

These simple examples demonstrate why process rewrite systems find a natural application in the interprocedural control-flow analysis of programs [35]. More details can be found, e.g., in [15].

By restricting the general form of the rewrite rules we obtain several subclasses of process rewrite systems. Let $\mathcal{S}$ (sequential process expressions) represent the subclass of general process expressions $\mathcal{G}$ that contains no parallel operator. Also, let $\mathcal{P}$ (parallel process expressions) represent the subclass of general process expressions $\mathcal{G}$ that contains no sequential operator. Let 1 be the class that contains only process constants and the empty process. Then, e.g., $(1, \mathcal{P})$-PRS is the subclass of PRS where every rule $E \xrightarrow{a} E'$ is restricted such that E is a process constant only and E' is an arbitrary parallel expression.

The complete PRS-hierarchy is depicted in Figure 2. This hierarchy is known to be strict with respect to strong bisimilarity and none of the classes in the hierarchy is Turing powerful since, e.g., the reachability problem is decidable even for the whole PRS class [37].

The reader may wonder what is the connection with the models of infinite-state systems introduced in previous sections? The answer is surprisingly simple. Most of the classes in Figure 2 correspond naturally to the well-known classes of processes like context-free processes, pushdown automata and Petri nets. Rules of the type $X \xrightarrow{a} Y.Z$ define the BPA class, rules of the type $X \xrightarrow{a} Y|Z$ correspond to BPP processes, rules like $X.Y \xrightarrow{a} Z.U$ characterize the PDA class (the correspondence is not straightforward and was proved in [11] by Caucal), rules of the type $X|Y \xrightarrow{a} Z|U$ are Petri net rules, and $X \xrightarrow{a} (Y|Z).U$ is a characteristic rule for the PA-processes of Baeten and Weijland [4].

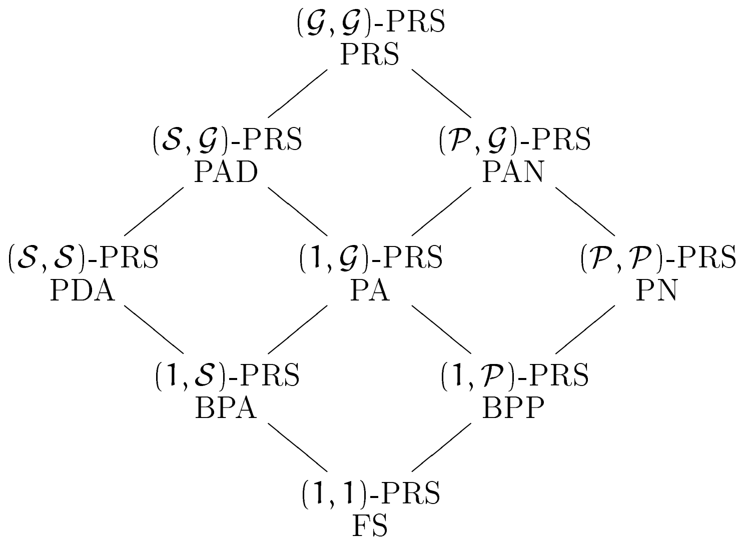


Fig. 2. The PRS-Hierarchy

Even the class of state-extended BPP has its place in the hierarchy. It lies between basic parallel processes (BPP) and Petri nets (PN) and its position is strict in both directions.

The study of decidability and complexity of bisimilarity checking problems for the classes from the PRS-hierarchy represents an active field of research; a summary of recent results is provided in [54].

7 Parallel Complexity

An intriguing question to ask about bisimulation is does it have an efficient *parallel* solution? The class NC contains those problems that can be solved in polylogarithmic time using a polynomial number of processors (in the size of the input). NC is generally regarded as the class of problems that have fast parallel solutions.

It is believed that P-complete problems cannot be in NC. A problem is in P if it can be solved by a deterministic Turing machine in polynomial time. A problem is P-complete if it is in the class P, and it is P-hard in the sense that any other problem in P is log-space reducible to it. A reduction is log-space if it uses at most a logarithmic amount of intermediate storage space.

Balcazar et al. [5] established the P-completeness of bisimulation checking on regular processes via a log-space reduction from the *Monotone Alternating Circuit Value Problem*. Despite this negative result, several parallel and distributed algorithms for deciding bisimulation equivalence of regular processes have been proposed that achieve non-trivial speedups in practice [71,32,49,8].

	strong/weak trace equivalence	*strong* bisimilarity	*weak* bisimilarity
FS	PSPACE-complete	P-complete	P-complete
BPA	Π^0_1-complete	$\in$2-EXPTIME	???
PDA	Π^0_1-complete	decidable	Σ^1_1-complete
BPP	Π^0_1-complete	PSPACE-complete	???
MSA	Π^0_1-complete	Π^0_1-complete	Σ^1_1-complete
PN	Π^0_1-complete	Π^0_1-complete	Σ^1_1-complete

Fig. 3. Summary of Complexity Results

8 Conclusions

We have offered a brief history of the computational complexity of bisimulation. Several comprehensive surveys about the subject, focusing on *infinite-state processes*, have been written (e.g., [44,31,30]), including a handbook chapter [9]. There is even now a project devoted to maintaining an up-to-date overview of the state of the art in this dynamic research topic [54].

The reader may have noticed the following trend about bisimulation equivalence: it is computationally easier to decide than language equivalence, regardless of the nature of the underlying process model, be it finite-state or infinite-state. It is interesting to search for an explanation to this computational dichotomy. Some insight can be gained by again noting that bisimulation is a much more discriminating equivalence than language equivalence, to the point where it is easier to decide. In particular, bisimilar states, for any symbol a, must lead to bisimilar states. The absence of this restriction on language-equivalent states in some sense forces one to determinize the automata in question to decide equivalence, a costly proposition indeed.

On the other hand weak bisimilarity is much harder to decide on infinite-state processes compared not only to (strong) bisimilarity but also to other (even weak) equivalences from van Glabbeek's spectrum such as the trace equivalences. Whenever a process formalism allows for a finite-state control unit, weak bisimilarity becomes highly undecidable (Σ^1_1-complete in the analytical hierarchy) [58] whereas, e.g., strong and weak trace equivalence remain on the first level of the arithmetical hierarchy (see [27]). A summary of these results is provided in Figure 3.

Finally, we ask what are the *practical ramifications* of the computational dichotomy? Happily, the answer is a positive one for computer scientists interested in bisimilarity, such as concurrency theorists and verification tool builders. In this case, one is confronted (at least for strong bisimilarity) with a tractable problem even for processes of a fairly expressive nature.

9 Acknowledgments

We would like to thank the anonymous referees for their helpful comments.

References

[1] S. Abramsky. The lazy lambda calculus. In D. A. Turner, editor, *Research Topics in Functional Programming*, pages 65–116. Addison-Welsey, Reading, MA, 1990.

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

[3] J.C.M. Baeten and R.J. van Glabbeek. Another look at abstraction in process algebra. In *Proceedings of the 14th International Colloquium on Automata, Languages and Programming (ICALP'87)*, volume 267 of *Lecture Notes in Computer Science*, pages 84–94. Springer-Verlag, 1987.

[4] J.C.M. Baeten and W.P. Weijland. *Process Algebra.* Number 18 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1990.

[5] J. Balcazar, J. Gabarro, and M. Santha. Deciding bisimilarity is P-complete. *Formal Aspects of Computing*, 4:638–648, 1992.

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

[7] J. Barwise and L. Moss. *Vicious Circles: On the Mathematics of Non-Wellfounded Phenomena.* CSLI Lecture Notes, No. 60, Stanford, CA, 1996.

[8] S. Blom and S. Orzan. A distributed algorithm for strong bisimulation reduction of state spaces. In L. Brim and O. Grumberg, editors, *Parallel and Distributed Model Checking (PDMC 2002).* Elsevier Science, Electronic Notes in Theoretical Computer Science 68 No. 4, 2002.

[9] O. Burkart, D. Caucal, F. Moller, and B. Steffen. Verification on infinite structures. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, chapter 9, pages 545–623. Elsevier Science, 2001.

[10] O. Burkart, D. Caucal, and B. Steffen. An elementary decision procedure for arbitrary context-free processes. In *Proceedings of the 20th International*

Symposium on Mathematical Foundations of Computer Science (MFCS'95), pages 423–433. Volume 969 of *Lecture Notes in Computer Science*, Springer-Verlag, 1995.

[11] D. Caucal. On the regular structure of prefix rewriting. *Theoretical Computer Science*, 106(1):61–86, 1992.

[12] S. Christensen, Y. Hirshfeld, and F. Moller. Bisimulation is decidable for basic parallel processes. In *Proceedings of the 4th International Conference on Concurrency Theory (CONCUR'93)*, volume 715 of *Lecture Notes in Computer Science*, pages 143–157. Springer-Verlag, 1993.

[13] S. Christensen, Y. Hirshfeld, and F. Moller. Decomposability, decidability and axiomatisability for bisimulation equivalence on basic parallel processes. In *Proceedings of the 8th Annual IEEE Symposium on Logic in Computer Science*, pages 386–396. IEEE, 1993.

[14] S. Christensen, H. Hüttel, and C. Stirling. A polynomial algorithm for deciding bisimilarity of normed context-free processes. *Information and Computation*, 12(2):143–148, 1995.

[15] J. Esparza and J.Knoop. An automata-theoretic approach to interprocedural data-flow analysis. In *Proceedings of the 2nd International Conference on Foundations of Software Science and Computation Structures (FOSSACS'99)*, volume 1578 of *Lecture Notes in Computer Science*, pages 14–30. Springer-Verlag, 1999.

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

[17] J. Hartmanis. Turing Award lecture: On computational complexity and the nature of computer science. *Communications of the ACM*, 37(10):37–43, October 1994.

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

[19] Y. Hirshfeld. Petri nets and the equivalence problem. In *Proceedings of the 7th Workshop on Computer Science Logic (CSL'93)*, volume 832 of *Lecture Notes in Computer Science*, pages 165–174. Springer-Verlag, 1994.

[20] Y. Hirshfeld. Bisimulation trees and the decidability of weak bisimulations. In *Proceedings of the 1st International Workshop on Verification of Infinite State Systems (INFINITY'96)*, volume 5 of *Electronic Notes in Theoretical Computer Science*. Springer-Verlag, 1996.

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

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

[23] C.A.R. Hoare. Communicating sequential processes. In *On the construction of programs – an advanced course*, pages 229–254. Cambridge University Press, 1980.

[24] H. B. Hunt, D. J. Rosenkrantz, and T. G. Szymanski. On the equivalence, containment, and covering problems for the regular and context-free languages. *Journal of Computer and System Sciences*, 12:222–268, 1976.

[25] H. Hüttel. Undecidable equivalences for basic parallel processes. In *Proceedings of the 2nd International Symposium on Theoretical Aspects of Computer Software (TACS'94)*, volume 789 of *Lecture Notes in Computer Science*, pages 454–464. Springer-Verlag, 1994.

[26] D.T. Huynh and L. Tian. On deciding readiness and failure equivalences for processes in Σ_2^{P}. *Information and Computation*, 117(2):193–205, 1995.

[27] P. Jančar. High undecidability of weak bisimilarity for Petri nets. In *Proceedings of Colloquium on Trees in Algebra and Programming (CAAP'95)*, volume 915 of *Lecture Notes in Computer Science*, pages 349–363. Springer-Verlag, 1995.

[28] P. Jančar. Undecidability of bisimilarity for Petri nets and some related problems. *Theoretical Computer Science*, 148(2):281–301, 1995.

[29] P. Jančar. Strong bisimilarity on basic parallel processes is PSPACE-complete. In *Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science (LICS'03)*, pages 218–227. IEEE Computer Society Press, 2003.

[30] P. Jančar and A. Kučera. Equivalence-checking with infinite-state systems: Techniques and results. In *Proceedings of the 29th Annual Conference on Current Trends in Theory and Practice of Informatics (SOFSEM'02)*, volume 2540 of *Lecture Notes in Computer Science*, pages 41–73. Springer-Verlag, 2002.

[31] P. Jančar and F. Moller. Techniques for decidability and undecidability of bisimilarity – an invited tutorial. In *Proceedings of the 10th International Conference on Concurrency Theory (CONCUR'99)*, volume 1664 of *Lecture Notes in Computer Science*, pages 30–45. Springer-Verlag, 1999.

[32] C. Jeong, Y. Kim, H. Kim, and Y. Oh. A faster parallel implementation of Kanellakis-Smolka algorithm for bisimilarity checking. In *Proceedings of the International Computer Symposium*, Tainan, Taiwan, 1998.

[33] A. Joyal, M. Nielsen, and G. Winskel. Bisimulation from open maps. *Information and Computation*, 127(2):164–185, 1996.

[34] P. C. Kanellakis and S. A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86(1):43–68, May 1990.

[35] J. Knoop. *Optimal Interprocedural Program Optimization: A New Framework and its Application*, volume 1428 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.

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

[37] R. Mayr. Process rewrite systems. *Information and Computation*, 156(1):264–286, 2000.

[38] R. Mayr. Weak bisimilarity and regularity of BPA is EXPTIME-hard. In *Proceedings of the 10th International Workshop on Expressiveness in Concurrency (EXPRESS'03)*, pages 160–143, 2003. To appear in ENTCS.

[39] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1980.

[40] R. Milner. A modal characterization of observable machine-behaviour. In *Proceedings of the 6th Colloquium on Trees in Algebra and Programming (CAAP'81)*, volume 112 of *Lecture Notes in Computer Science*, pages 25–34. Springer-Verlag, 1981.

[41] R. Milner. *Communication and Concurrency*. International Series in Computer Science. Prentice Hall, 1989.

[42] R. Milner. Elements of interaction — Turing Award lecture. *Communications of the ACM*, 36(1):78–89, January 1993.

[43] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, Parts I and II. *Information and Computation*, 100, 1992.

[44] F. Moller. Infinite results. In *Proceedings of the 7th International Conference on Concurrency Theory (CONCUR'96)*, volume 1119 of *Lecture Notes in Computer Science*, pages 195–216. Springer-Verlag, 1996.

[45] F. Moller and S. A. Smolka. On the computational complexity of bisimulation. *ACM Computing Surveys*, 27(2):287–289, June 1995.

[46] F. Moller and S. A. Smolka. On the computational complexity of bisimulation, redux. In *PCK50: Principles of Computing and Knowledge: Paris C. Kanellakis Memorial Workshop*, San Diego, CA, June 2003. ACM.

[47] R. Paige and R. E. Tarjan. Three partition refinement algorithms. *SIAM Journal of Computing*, 16(6):973–989, December 1987.

[48] D. M. R. Park. Concurrency and automata on infinite sequences. In *Proceedings of the 5th G.I. Conference on Theoretical Computer Science*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer-Verlag, 1981.

[49] S. Rajasekaran and I. Lee. Parallel algorithms for relational coarsest partition problems. *IEEE Transactions on Parallel and Distributed Systems*, 9(7), July 1998.

[50] J.M. Rutten. Universal coalgebra: A theory of systems. *Theoretical Computer Science*, 249(1):3–80, 2000.

[51] K. Segerberg. An essay in classical modal logic. *Filosofiska Studier*, 13:301–322, 1971.

[52] G. Senizergues. Decidability of bisimulation equivalence for equational graphs of finite out-degree. In *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science*, pages 120–129. IEEE, 1998.

[53] G. Senizergues. L(A)=L(B)? Decidability results from complete formal systems. *Theoretical Computer Science*, 251(1-2):1–166, January 2001.

[54] J. Srba. Roadmap of infinite results. *Bulletin of the European Association for Theoretical Computer Science (Columns: Concurrency)*, 78:163–175, October 2002. Updated online version: http://www.brics.dk/~srba/roadmap.

[55] J. Srba. Strong bisimilarity and regularity of basic parallel processes is PSPACE-hard. In *Proceedings of the 19th International Symposium on Theoretical Aspects of Computer Science (STACS'02)*, volume 2285 of *Lecture Notes in Computer Science*, pages 535–546. Springer-Verlag, 2002.

[56] J. Srba. Strong bisimilarity and regularity of Basic Process Algebra is PSPACE-hard. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming (ICALP'02)*, pages 716–727. Volume 2380 of *Lecture Notes in Computer Science*, Springer-Verlag, 2002.

[57] J. Srba. Undecidability of weak bisimilarity for pushdown processes. In *Proceedings of the 13th International Conference on Concurrency Theory (CONCUR'02)*, volume 2421 of *Lecture Notes in Computer Science*, pages 579–593. Springer-Verlag, 2002.

[58] J. Srba. Completeness results for undecidable bisimilarity problems. In *Proceedings of the 5th International Workshop on Verification of Infinite-State Systems (INFINITY'03)*, pages 9–22, 2003. To appear in ENTCS.

[59] C. Stirling. Games for bisimulation and model checking. In *Notes for Mathfit Workshop on Finite Model Theory*, University of Wales, Swansea, July 1996.

[60] C. Stirling. Decidability of bisimulation equivalence for pushdown processes. Research Report EDI-INF-RR-0005, School of Informatics, Edinburgh University, January 2000.

[61] C. Stirling. Decidability of DPDA equivalence. *Theoretical Computer Science*, 255(1-2):1–31, March 2001.

[62] C. Stirling. Decidability of weak bisimilarity for a subset of basic parallel processes. In *Proceedings of the 4th International Conference on Foundations of Software Science and Computation Structures (FOSSACS'01)*, volume 2030 of *Lecture Notes in Computer Science*, pages 379–393. Springer-Verlag, 2001.

[63] C. Stirling. Deciding DPDA equivalence is primitive recursive. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming (ICALP'02)*, volume 2380 of *Lecture Notes in Computer Science*, pages 821–832. Springer-Verlag, 2002.

[64] W. Thomas. On the Ehrenfeucht-Fraïssé game in theoretical computer science (extended abstract). In *Proceedings of the 4th International Joint Conference CAAP/FASE, Theory and Practice of Software Development (TAPSOFT'93)*, volume 668 of *Lecture Notes in Computer Science*, pages 559–568. Springer-Verlag, 1993.

[65] D. Turi and J. Rutten. On the foundations of final coalgebra semantics: non-well-founded sets, partial orders, metric spaces. *Mathematical Structures in Computer Science*, 8(5):481–540, 1998.

[66] J. van Bentham. Correspondence theory. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, chapter II.4, pages 167–247. Kluwer Academic Publishers, 1984.

[67] R.J. van Glabbeek. *Comparative Concurrency Semantics and Refinement of Actions.* PhD thesis, CWI/Vrije Universiteit, 1990.

[68] R.J. van Glabbeek. The linear time—branching time spectrum. In *Proceedings of the 1st International Conference on Theories of Concurrency: Unification and Extension (CONCUR'90)*, volume 458 of *Lecture Notes in Computer Science*, pages 278–297. Springer-Verlag, 1990.

[69] R.J. van Glabbeek and W.P. Weijland. Branching time and abstraction in bisimulation semantics. *Information Processing Letters*, 89:613–618, 1989.

[70] R.J. van Glabbeek and W.P. Weijland. Branching time and abstraction in bisimulation semantics. *Journal of the ACM*, 43(3):555–600, 1996.

[71] S. Zhang and S. A. Smolka. Efficient parallelization of equivalence checking algorithms. In M. Diaz and R. Groz, editors, *Proceedings of the 5th International Conference on Formal Description Techniques*, pages 133–146, October 1992.