FI MU

Faculty of Informatics
Masaryk University

Pre-proceedings

MFCS'98 Workshop on Concurrency

August 27-29, 1998
Brno, Czech Republic

Petr Jančar and Mojmír Křetínský (Editors)

FI MU Report Series **FIMU-RS-98-06**

 July 1998

This pre-proceedings constitute preliminary versions of the papers. The final versions will appear as a volume in ENTCS (Electronic Notes in Theoretical Computer Science, Elsevier) at URL http://www.elsevier.nl/locate/entcs

Foreword

This volume contains the pre-proceedings of the *MFCS'98 Workshop on Concurrency*, which took place in Brno, Czech Republic, 27–29 August 1998, as a satellite event of MFCS'98, the 23rd international symposium on Mathematical Foundations of Computer Science.
The revised (and full) versions of the contributions should appear as a volume of *Electronic Notes of Theoretical Computer Science*; cf. URL `http://www.elsevier.nl/locate/entcs`.
The idea of the Workshop was initiated by Jozef Gruska, one of the co-chairs of MFCS'98. The Workshop became one of several satellite events of MFCS'98; its aim was to provide a special forum for researchers in the area of concurrency participating in MFCS'98 but also to attract further interested researchers.
The call of papers suggested the topics like decidability and complexity of testing behavioural equivalences, model checking and other verification problems for various models of (concurrent) systems, models for concurrency, practical tools for modelling and verification of concurrent systems, verification of infinite-state processes, but it did not limit the submissions to these topics.
According to evaluations of the programme committee members, assisted by further referees, 17 submissions (out of 24) have been selected for presentation at the Workshop.
Their (preliminary) written versions are contained in this volume, accompanied by the texts sent to us by the two invited speakers at the Workshop, Javier Esparza (Munich) and Faron Moller (Uppsala).
We would like to thank very much to Ahmed Bouajjani (Grenoble), Julian Bradfield (Edinburgh), Wilfried Brauer (Munich), Mogens Nielsen (Aarhus) and Colin Stirling (Edinburgh) who kindly accepted the participation in the programme committee and the work connected with the submission evaluation.
We also wish to thank to Jan Staudek, the chair of the MFCS'98 organising committee, as well as to the local organising committee of the Workshop, who helped with its realization.
We also acknowledge the partial support of the Grant Agency of the Czech Republic to the Workshop – via grant No. 201/97/0456.

Brno, July 1998

Petr Jančar and Mojmír Křetínský
Programme Committee Co-chairs

Programme Committee

Ahmed Bouajjani, Grenoble (F)
Julian Bradfield, Edinburgh (UK)
Wilfried Brauer, Munich (D)
Petr Jančar (co-chair), Ostrava (CZ)
Mojmír Křetínský (co-chair), Brno (CZ)
Mogens Nielsen, Aarhus (DK)
Colin Stirling, Edinburgh (UK)

Organising Committee

Ivana Černá, Brno (CZ), Antonín Kučera, Brno (CZ), Jiří Srba, Brno (CZ)

Contents

Invited Talks

Contributed Papers

Verification with unfoldings

Javier Esparza

Institut für Informatik
Technische Universität München
esparza@informatik.tu-muenchen.de

The automatic verification of finite state systems suffers from the explosion of states caused by the many possible permutations of concurrent events. Unfoldings are a verification technique that avoids this explosion by disregarding the order of concurrent events. It belongs to the group of so-called partial-order methods for model checking, which also contains Valmari's stubborn sets (implemented in the PROD tool), Godefroid's sleep sets (implemented in Holzman's SPIN), and others.

Petri nets are a natural model for the unfolding approach, since they make concurrent events explicit, but the technique can be equally well applied to communicating automata. The behaviour of the Petri net is captured by unfolding of the net into an infinite acyclic occurrence net. The unfolding operation is similar to the unwinding of a finite automaton into an infinite acyclic automaton, but retains the concurrency aspects of the net.

The use of unfoldings for automatic verification was first proposed by K.L. McMillan in his Ph. D. Thesis, where he showed that the infinite unwinding can be terminated when it contains full information about the reachable states of the original net, even though the states are not represented explicitly. In this sense, this prefix of the infinite unwinding can be seen as a compact encoding of the state space. Since the permutations of concurrent events are not enumerated, the prefix can be much smaller than the state graph of the system. A weakness of McMillan's original proposal was that the prefix could also be larger (even exponentially larger) than the state graph. This problem was solved by Esparza, Roemer, and Vogler by means of an improved criterion for termination.

Different authors have defined, implemented and tested algorithms which use unfoldings to efficiently solve a number of verification problems: deadlock detection, reachability, concurrency of events, model-checking for both branching and linear time logics, and conformance between trace structures. Most of these algorithms are part of PEP, a modelling and verification tool developed at the University of Oldenburg and the Technical University of Munich.

Unfolding techniques have been applied to problems in different areas, such as design of asynchronous circuits, verification of protocols and manufacturing systems, management of telecommunication networks, and others.

The talk presents the basic ideas of the unfolding technique, compares it with others, and provides performance statistics on some case studies. It finishes with pointers to further information and available software.

Pushdown Automata, Multiset Automata, and Petri Nets

Faron Moller
Computing Science Department
Uppsala University

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

Abstract

We consider various classes of automata generated by simple rewrite transition systems. These classes are defined by two natural hierarchies, one given by interpreting catenation of symbols in the rewrite system as sequential composition, and the other by interpreting catenation as parallel composition. In this way we provide natural definitions for commutative (parallel) context-free automata, multiset (parallel push-down) automata, and Petri nets.

1 Introduction

Consider a ***context-free grammer*** (CFG) in ***Greibach normal form*** (GNF), for example as given by the following three rules.

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

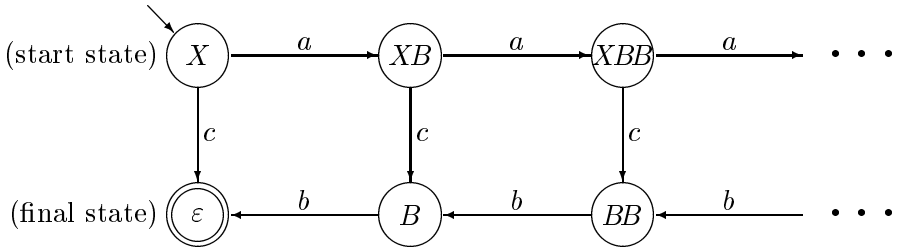
Such a grammar consists of

a set of ***variables***	V	$= \{X, B\};$
a set of ***alphabets labels***	Σ	$= \{a, b, c\};$ and
a set of ***production rules***	P	$= \{\ X \longrightarrow aXB,$
		$\quad X \longrightarrow c,$
		$\quad B \longrightarrow b\ \}.$

As we are concerned only with GNF grammars, we shall always write the production rules with the label on top of the arrow. Also associated with such a grammar is an ***initial variable*** X, and the ***context-free language*** (CFL) generated by (the initial variable of) the grammar, in this case

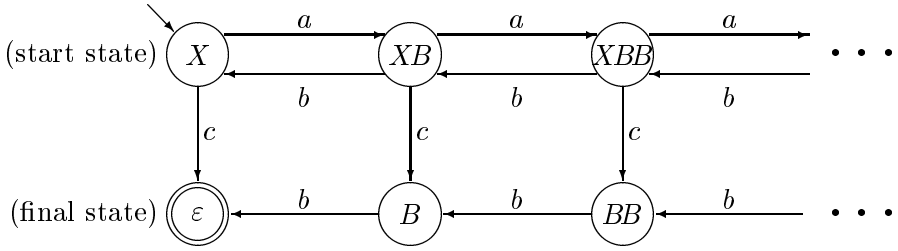
$$L(X) \;=\; \{a^k c b^k \;:\; k \geq 0\}.$$

Restricting to ***leftmost derivations*** gives rise to the following ***automaton***.



Such grammars make up Bergstra and Klop's ***Basic Process Algebra*** (BPA) [4] and their automata are referred to as BPA processes. They are also instances of Caucal's ***Rewrite Transition Systems*** [8].

We can also interpret catenation of variables as *"parallel"* rather than *"sequential"* composition, by reading sequences of variables modulo commutativity of catenation. Thus for example, $XBB = BXB = BBX$. Under this interpretation, the above grammar gives rise to the following automaton.



Such an interpretation gives rise to Christensen's ***Basic Parallel Processes*** (BPP) [10]. Note that its language, defined in the natural way as the sequence of labels on paths leading from the start state to the final state, is generally different from the language of the sequential automaton (which itself coincides naturally with the language of the CFG). In fact, its language need *not* even be context-free. For example, the BPP given by the grammar

$$\begin{array}{lll} X \stackrel{a}{\longrightarrow} BCX & X \stackrel{b}{\longrightarrow} ACX & X \stackrel{c}{\longrightarrow} ABX \\ X \stackrel{a}{\longrightarrow} BC & X \stackrel{b}{\longrightarrow} AC & X \stackrel{c}{\longrightarrow} AB \\ A \stackrel{a}{\longrightarrow} \varepsilon & B \stackrel{b}{\longrightarrow} \varepsilon & C \stackrel{c}{\longrightarrow} \varepsilon \end{array}$$

generates the non-CFL consisting of the strings of $\{a, b, c\}^*$ containing an equal number of a's, b's and c's.

In the following, we shall generalise these processes and consider various questions regarding their equivalence checking problems.

2 Rewrite Transition Systems

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

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

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

This notion of a labelled transition system differs from the standard definition of a finite-state automata only in that the set of states need not be finite, and final states must not have any outgoing transitions.

Definition 2.2 A ***sequential labelled rewrite transition system*** is a tuple $\langle V, \Sigma, P, \alpha_0, F \rangle$ where

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

A ***parallel labelled rewrite transition system*** is defined precisely as above, except that the elements of V^* are read modulo commutativity of catenation, which is thus interpretted as parallel, rather than sequential, composition.

We shall freely extend the transition relation $\longrightarrow$ homomorphically to finite sequences of actions $w \in \Sigma^*$ so as to write $\alpha \xrightarrow{\varepsilon} \alpha$ and $\alpha \xrightarrow{aw} \beta$ whenever $\alpha \xrightarrow{a} \gamma \xrightarrow{w} \beta$ 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 $\alpha_0 \xrightarrow{w} \alpha X \beta$.

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

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

In the remainder of this section, we explain the classes of transition systems which are represented in this table, working upwards starting with the most restrictive classes. In drawing labelled transition systems, we shall continue our trend of indicating initial states by short arrows, and indicating finalstates by double circles.

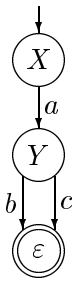
FSA represents the class of ***finite-state automata***. Clearly if the rules are restricted to be of the form $A \xrightarrow{a} B$ or $A \xrightarrow{a} \varepsilon$ with $A, B \in V$, then the reachable states of both the sequential and parallel transition systems will be a subset of the finite set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

Example 1 *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.*

$X \xrightarrow{a} Y$

$Y \xrightarrow{b} \varepsilon$

$Y \xrightarrow{c} \varepsilon$

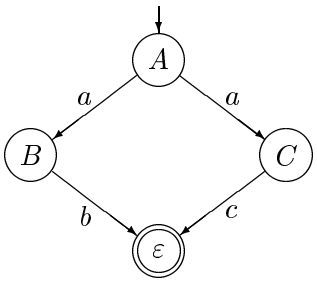


$A \xrightarrow{a} B$

$A \xrightarrow{a} C$

$B \xrightarrow{b} \varepsilon$

$C \xrightarrow{c} \varepsilon$



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.*

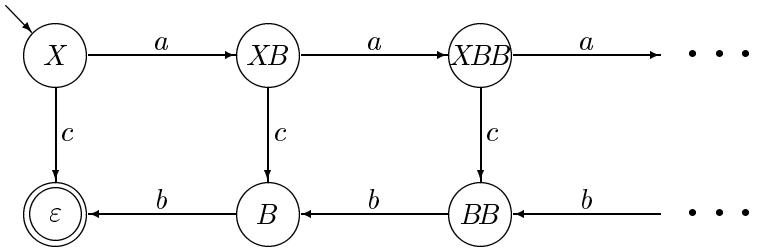
As indicated above, BPA represents the class of ***Basic Process Algebra*** processes of Bergstra and Klop [4], which are the transition systems associated with GNF context-free grammars in which only left-most derivations are permitted.

Example 2 *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 automata recognises the context-free language $\{ a^n c b^n \ : \ n \geq 0 \}$.

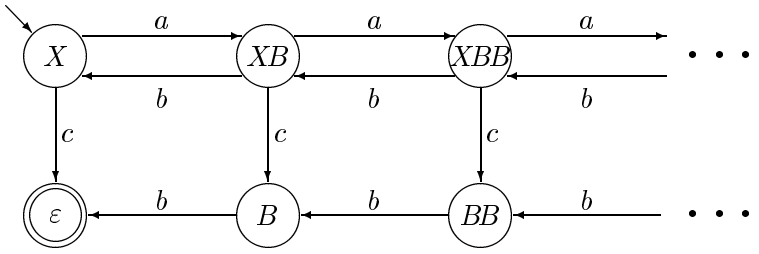
Also as indicated above, BPP represents the class of ***Basic Parallel Processes*** introduced by Christensen [10] as a parallel analogy to BPA, and are defined by the transition systems associated with GNF context-free grammars in which arbitrary grammar derivations are permitted.

Example 3 *The type 2 rewrite system from Example 2 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 automata 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 ***push-down automata*** which accept on empty stack. To present such PDA as a restricted form of rewrite system, we first assume that the variable set V is partitioned into disjoint sets Q (finite control states) and , (stack symbols). The rewrite rules are then of the form $pA \xrightarrow{a} q\beta$ with $p, q \in Q$, $A \in$, and $\beta \in$, *, 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 [8] 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 n 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 $Q = \{\, p_\alpha \;:\; \alpha \in V^* \text{ and } \mathrm{length}(\alpha) < n \,\}$ and , $= V \cup \{\, Z_\alpha \;:\; \alpha \in V^* \text{ and } \mathrm{length}(\alpha) \leq n \,\}$. 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 non-final 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 2.3 *Every sequential labelled rewrite transition system can be represented (up to the labelling of states) by a PDA transition system.*

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 3 must have at least 2 control states: this transition system is not represented by any BPA.

Example 4 *The BPP transition system of Example 3 is given by the following sequential rewrite system.*

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

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

$$p_X Z_\varepsilon \xrightarrow{a} p_X Z_B \qquad p_X Z_{BB} \xrightarrow{a} p_X B Z_{BB} \qquad p_B Z_\varepsilon \xrightarrow{b} p_\varepsilon$$
$$p_X Z_\varepsilon \xrightarrow{c} p_\varepsilon \qquad p_X Z_{BB} \xrightarrow{b} p_X Z_B \qquad p_B Z_B \xrightarrow{b} p_B Z_\varepsilon$$
$$p_X Z_{BB} \xrightarrow{c} p_B Z_B \qquad p_B Z_{BB} \xrightarrow{b} p_B Z_B$$
$$p_B B \xrightarrow{b} p_B$$
$$p_X Z_B \xrightarrow{a} p_X Z_{BB} \qquad p_X B \xrightarrow{a} p_X BB$$
$$p_X Z_B \xrightarrow{b} p_X Z_\varepsilon \qquad p_X B \xrightarrow{b} p_X$$
$$p_X Z_B \xrightarrow{c} p_B Z_\varepsilon \qquad p_X B \xrightarrow{c} p_B$$

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

$$pZ \xrightarrow{a} pBZ \qquad pB \xrightarrow{a} pBB \qquad qZ \xrightarrow{c} q$$
$$pZ \xrightarrow{c} q \qquad pB \xrightarrow{b} p \qquad qB \xrightarrow{b} q$$
$$pB \xrightarrow{c} pBB$$

MSA represents the class of ***multiset automata***, which can be viewed as *"parallel"* or *"random-access"* push-down automata. They are defined as above except that they have random access capability to the stack. Thus a MSA transition rule $pA \xrightarrow{a} q\beta$ with $p, q \in Q$, $A \in$, and $\beta \in$, *, 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 5 *The BPA transition system of Example 2 is isomorphic to that given by the following MSA with initial state pX.*

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

Note that when the stack alphabet has only one element, PDA and 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 2 must have at least 2 control states: this transition system is not represented by any BPP.

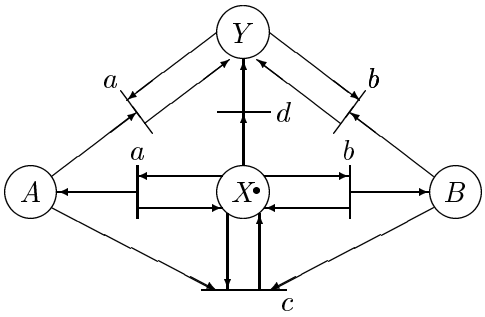
PN represents the class of (finite, labelled, weighted place/transition) ***Petri nets***, as is evident by the following interpretation of unrestricted parallel rewrite systems. The variable set V represents the set of places of the Petri net, and each rewrite rule $\alpha \xrightarrow{a} \beta$ represents 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 6 *The following unrestricted parallel rewrite system with initial state X and final state Y*

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

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



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

Although in the sequential case, PDA constitutes a normal form for unrestricted rewrite transition systems, we shall see that this result does not hold in the parallel case.

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 3.1 $L(\alpha) = \{ w \in \Sigma^* : \alpha \xrightarrow{w} \beta \text{ for some } \beta \in F \}$, and $L(T) = L(\alpha_0)$. α and β are ***language equivalent***, written $\alpha \sim_L \beta$, iff they generate the same language: $L(\alpha) = L(\beta)$.

Thus, for example, the class of languages generated by FSA are precisely the (ε-free) regular languages; and the class of 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 3.2 We define the ***norm*** of any state α of a labelled transition system, written $\mathrm{norm}(\alpha)$, to be the length of a shortest rewrite transition sequence which takes α to a final state, that is, the length of a shortest word in $L(\alpha)$. By convention, we define $\mathrm{norm}(\alpha) = \infty$ if there is no sequence of transitions from α to a final state, that is, $L(\alpha) = \emptyset$. The transition system is ***normed*** iff every reachable state α has a finite norm.

Note that, due to our assumption following Definition 2.2 on the accessibility of all 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 3.3 *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 3.4 T is ***deterministic*** iff for every reachable state α and every label a there is at most one state β such that $\alpha \stackrel{a}{\longrightarrow} \beta$.

For example, the two finite-state automata presented in Example 1 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 1 which generate the same language $\{\, ab,\, ac \,\}$ yet demonstrate different deadlocking capabilities due to the nondeterministic behaviour exhibitted by the second transition system. Many finer equivalences have been proposed, with ***bisimulation equivalence*** being perhaps the finest behavioural equivalence studied. (Note that we do not consider here any so-called 'true concurrency' equivalences such as those based on partial orders.) Bisimulation equivalence was defined by Park [42] and used to great effect by Milner [35, 36]. Its definition, in the presence of final states, is as follows.

Definition 3.5 A binary relation $\mathcal{R}$ on states of a transition system is a ***bisimulation*** iff whenever $\langle \alpha, \beta \rangle \in \mathcal{R}$ we have that

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

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

Lemma 3.6 $\sim = \bigcup \{ \mathcal{R} : \mathcal{R}$ *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 [46]. Starting with a pair of states $\langle \alpha, \beta \rangle$, the two players alternate moves according to the following rules.

1. If exactly one of the pair of states is a final state, then player I is deemed to be the winner. Otherwise, player I chooses one of the states and makes some transition from that state (either $\alpha \xrightarrow{a} \alpha'$ or $\beta \xrightarrow{a} \beta'$). If this proves impossible, due to both states being terminal, then player II is deemed to be the winner.
2. Player II must respond to the move made by player I by making an identically-labelled transition from the other state (either $\beta \xrightarrow{a} \beta'$ or $\alpha \xrightarrow{a} \alpha'$). If this proves impossible, then player I is deemed to be the winner.
3. The play then repeats itself from the new pair $\langle \alpha', \beta' \rangle$. If the game continues forever, then player II is deemed to be the winner.

The following result is then immediately evident.

Fact 3.7 $\alpha \sim \beta$ *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 3.8 *If* $\alpha \sim \beta$ *and* $\alpha \xrightarrow{w} \alpha'$ *with* $w \in \Sigma^*$, *then* $\beta \xrightarrow{w} \beta'$ *such that* $\alpha' \sim \beta'$.

Corollary 3.9 *If* $\alpha \sim \beta$ *then* $\alpha \sim_L \beta$.

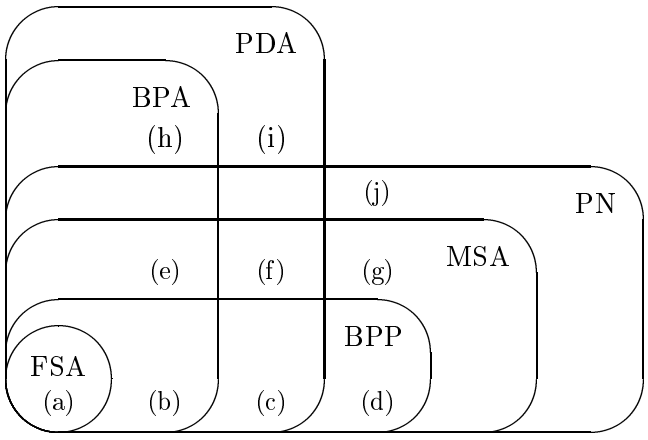
Apart from being the fundamental notion of equivalence for several process algebraic formalisms, bisimulation equivalence has several pleasing mathematical properties, not least of which being that it is decidable over classes of transition systems for which all other common equivalences, including language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

Lemma 3.10 *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 3.9, $\sim_L$ 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 push-down automata, for which a lengthy proof was only recently demonstrated by Sénizergues [44].

4 Expressivity Results

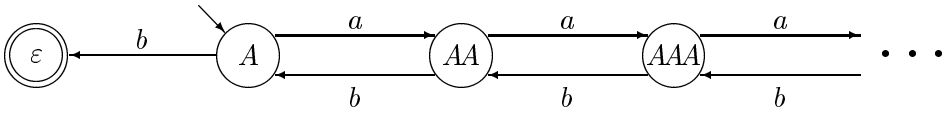
Our hierarchy from above gives us the following classification of processes.



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.

(a) The first transition system in example 1 provides a normed deterministic FSA.

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



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

(c) Examples 3 and 4 provide a transition system which can be described by both a BPP (Example 3) and a PDA (Example 4). 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 3 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 3 has two such non-bisimilar derived states, namely XB^{k-1} and B^k where $k = \text{norm}(\alpha)$.

(d) The following BPP with initial state X

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

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

(e) Examples 2 and 5 provide a transition system which can be described by both a BPA (Example 2) and a MSA (Example 5). However, the context-free language which it generates, $\{a^ncb^n \ : \ n \geq 0\}$, cannot be generated by any BPP, so this transition system is not even language equivalent to any BPP. To see this, suppose that $L(X) = \{a^ncb^n \ : \ n \geq 0\}$ for some BPP state X. (As the process has 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^kcb^k:

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

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

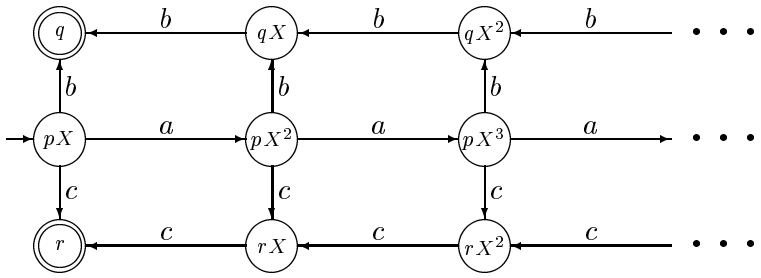
$$X \xrightarrow{a^k} Y\alpha \xrightarrow{b^i} Y \xrightarrow{c} \beta \xrightarrow{b^{k-i}} \varepsilon$$

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

(f) The following PDA with initial state pX

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

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

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

generates the language $\{\, a^n b^k c^n \;:\; 0 < k \leq n \,\}$, and hence cannot be language equivalent to any 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

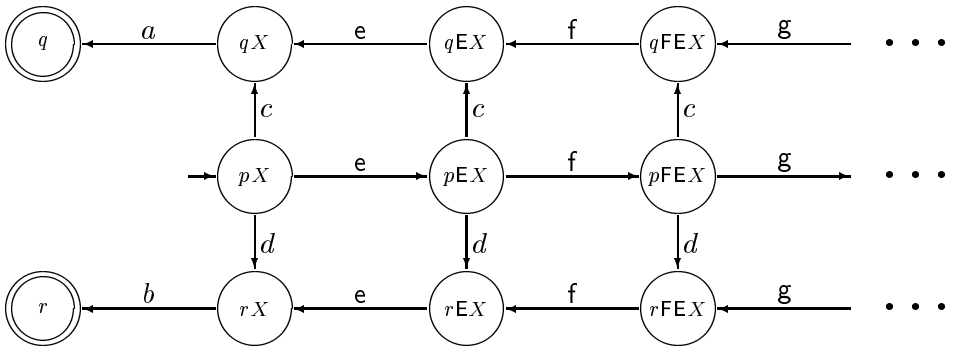
$$X \xrightarrow{a} XA \qquad X \xrightarrow{b} XB \qquad X \xrightarrow{c} \varepsilon \qquad A \xrightarrow{a} \varepsilon \qquad B \xrightarrow{b} \varepsilon$$

generates the language $\{\, wcw^{\mathrm{R}} \;:\; w \in \{a,b\}^* \,\}$ and hence is not language equivalent to any PN [43].

(i) The following PDA with initial state pX

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

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 $\{\, wcw^{\mathrm{R}}a,\ wcw^{\mathrm{R}}b \ : \ w \in \{a, b\}^* \,\}$ 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 6 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 [18], 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 isn't already given by the Lemma, then we can merely extend the initial segment to the next multiple of the length of the cycle given by the Lemma, and use this multiple as the cycle length.) Considering now an indefinite sequence of b-transitions from $q\alpha$, 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

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

Consider then the following sequence of transitions.

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

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

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

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

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

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

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

Theorem 4.1 *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.*

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 [20] 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 (Moore [40]). 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 (Paige and Tarjan [41], Kanellakis and Smolka [33]).

The first relevant result related to infinite-state automata is the undecidability of language equivalence between context-free automata BPA (Bar-Hillel, Perles and Shamir [3]). Groote and Hüttel [17] extend this undecidability result to all of the equivalences in van Glabbeek's catalogue of equivalences [15] except for bisimulation. Baeten, Bergstra and Klop [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 [37, 38] were soon recorded, notably by Caucal [7], Hüttel and Stirling [26], and Groote [16]. Huynh and Tian [27] demonstrate that this problem has a complexity of Σ_2^{P} by providing a nondeterministic algorithm which relies on an NP oracle; Hirshfeld, Jerrum and Moller [21, 22] 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 [34], and the singly-exponential algorithm of Caucal [9]. A generally more efficient, though worst-case exponential, algorithm is presented by Hirshfeld and Moller [24]. Finally, Christensen, Hüttel and Stirling [13, 14] demonstrate the general problem to be decidable, whilst Burkart, Caucal and Steffen [5] provide an elementary decision procedure.

For the case of commutative context-free automata BPP, we get similar results. Hirshfeld [19] demonstrates the undecidability of language equivalence, and Hüttel [25] extends this undecidability result to all of van Glabbeek's equivalences except bisimilarity. Christensen, Hirshfeld and Moller [11, 12] demonstrate the decidability of bisimilarity, first for the normed case and then in the general case; and Hirshfeld, Jerrum and Moller [23]

provide a polynomial-time algorithm for the normed case.

For PDA, we note the recent positive solution of Sénizergues [44] 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 [47] of the decidability of bisimilarity over normed PDA. The former proof is enormously long (exceeding 70pp in its full, as yet unpublished form [45]); 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 [28, 29] demonstrates the undecidability of bisimilarity for Petri nets, and this result is refined in [39] 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, Caucal and Steffen [5] 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 [48] consider the regularity checking problem for Petri nets with respect to language (and trace) equivalence; and Esparza, Jančar and Moller [32, 30, 31] 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. To view the myriad of results, look to Esparza's overview paper [6].

References

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

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

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

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

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

[6] O. Burkart and Javier Esparza (1997). More infinite results. Proceedings of Infinity'97. *Electronic Notes in Theoretical Computer Science* **5**.

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

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

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

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

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

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

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

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

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

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

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

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

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

[20] Y. Hirshfeld (1997). Unpublished.

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

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

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

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

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

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

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

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

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

[30] P. Jančar and J. Esparza (1996). Deciding finiteness of Petri nets up to bisimulation. Proceedings of ICALP'96, *Lecture Notes in Computer Science* **1099**:478–489.

[31] P. Jančar, J. Esparza and F. Moller (1997). Petri nets and regular processes. Submitted.

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

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

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

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

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

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

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

[39] F. Moller (1996). Infinite results. Proceedings of CONCUR'96, *Lecture Notes in Computer Science* **1119**:195–216.

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

[41] R. Paige and R.E. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing* **16**:937–989, 1987.

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

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

[44] G. Sénizergues (1997). The Equivalence Problem for Deterministic Pushdown Automata is Decidable. Proceedings of ICALP'97, *Lecture Notes in Computer Science* **1256**:671–681.

[45] G. Sénizergues (1997). L(A)=L(B)? Technical Report, LaBRI, Université Bordeaux I, report nr.1161-97.

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

[47] C. Stirling (1996). Decidability of bisimulation equivalence for normed pushdown processes. Proceedings of CONCUR'96, *Lecture Notes in Computer Science* **1119**:217–232.

[48] R. Valk and G. Vidal-Naquet (1981). Petri nets and regular languages. *Journal of Computer and System Sciences* **23**(3):299–325.

Queues as Processes

Olaf Burkart*
Lehrstuhl Informatik V
Universität Dortmund
44221 Dortmund, Germany
burkart@ls5.cs.uni-dortmund.de

Infinite-state systems, rewrite systems, queues, processes, reachability analysis, bisimulation, model checking.

Abstract

Rewrite systems have successfully been used for the description of infinite-state systems. According to the interpretation of words as either sequences or multisets rewrite systems may describe classes of transition systems like e.g. BPA, PDA, BPP or Petri Nets. In this paper we introduce a new hierarchy of processes obtained by considering rewrite systems together with a FIFO-like rewrite rule. We investigate the reachability, bisimulation and model checking problems for these processes.

1 Introduction

Recently, the use of rewrite systems for the description of infinite-state systems has blossomed within the concurrency theory community. One of the main motivations for their study has been to provide for a unified framework in which various well-known classes of transition systems first defined in the theory of process algebras or Petri nets can be concisely expressed. Moreover, this approach provides a clear link between well-studied classes of formal languages and families of transition systems on the other side. In one direction well-known decidability results from formal language theory can be used to answer decidability question about classes of transition systems, while in the other direction the more general viewpoint of considering graphs as generators of languages opens a new field of research in formal language theory.

Up to now rewrite systems have been used to describe classes of *sequential*, as well as *parallel* infinite-state processes leading to a Chomsky-like hierarchy. This taxonomy has attracted much research aimed at systematically clarifying the boundary between decidability and undecidability within the considered hierarchies for such divers problems as

*This research was supported by the DFG under grant STE 537/9-1.

reachability, bisimulation equivalence or model checking. Most of the results obtained are based on intriguing decomposition techniques, as well as interesting reductions to known undecidable problems and are summarized in [Mol96, Esp97, BE97].

In this paper we introduce a new third hierarchy of process classes. The main characteristic of these processes is a queue-like behaviour resulting from a new meta-rule which extends the given rewrite systems. Similar to the sequential and parallel cases this leads to a Chomsky-like hierarchy with three interesting classes, Basic Queue Processes (BQP), Finitely Controlled Queue Processes (FCQP) and Queue Processes (QP), for which we will consider the reachability, bisimulation equivalence, and model checking problem.

Queues are the model of choice in the design of e.g. round robin schedulers. The typical behaviour of such a scheduler operating on a queue of nondeterministic processes is (1) take the process from the head of the queue, (2) let it evolve for one transition, and (3) put the resulting successor state back to the end of the queue. The classes BQP, FCQP, and QP extend this basic behaviour by taking additional context into account, and by interpreting the successor state again as a queue.

The remainder of the paper is now organized as follows. In Section 2 we introduce rewrite systems coupled with queue-like behaviour which will be utilized to model infinite-state systems. Subsequently, we investigate the reachability problem for these processes in Section 3, whereas Section 4 addresses their bisimulation problems. Finally, we consider the model checking problem for the modal μ-calculus and queue processes in Section 5, while Section 6 contains our conclusions.

2 Models

Although a variety of semantic models exist for concurrent systems most can be interpreted as edge-labelled directed graphs whose vertices represent the states of the system, and whose edges describe the possible state transitions. Labels on the edges then represent the action or event that occurs.

Definition 2.1 *A* labelled transition system $\mathcal{T}$ *is a triple* $(\mathcal{S}, \Sigma, \{ \stackrel{a}{\rightarrow} \}_{a \in \Sigma})$ *where* $\mathcal{S}$ *is the set of states,* Σ *is the set of labels (or actions), and* $\stackrel{a}{\rightarrow} \subseteq \mathcal{S} \times \mathcal{S}$, $a \in \Sigma$ *are the transition relations.*

A transition system is rooted *if it has a distinguished initial state. A state* s'' *is reachable from* s' *if there exists a path* $s' = s_1 \stackrel{a_1}{\rightarrow} \ldots \stackrel{a_{n-1}}{\rightarrow} s_n = s''$.

Over the last decade a whole plethora of formalisms which may finitely represent infinite-state transition systems have been investigated. Here we follow the example set by Caucal [Cau92] and later by Stirling and Moller [Sti96, Mol96] who used rewrite systems to classify important classes of infinite-state transition systems.

Definition 2.2 (Labelled Rewrite Systems)
A labelled rewrite system *is a triple* $\mathcal{R} = (V, \Sigma, R)$ *where* V *is an* alphabet, Σ *is a set of* labels, *and* $R \subseteq V^* \times \Sigma \times V^*$ *is a finite set of* rewrite rules.

We use uppercase letters $A, B, C, \ldots$ to denote nonterminals of V, and lower case greek letters $\alpha, \beta, \gamma, \ldots$ to denote words over V. Moreover, we write ϵ for the empty word.

In the sequel, a rewrite rule $(\alpha_1, a, \alpha_2) \in R$ where $\alpha_i \in V^*$ is also written as $\alpha_1 \stackrel{a}{\to} \alpha_2$. We will denote a rewrite system simply by R if V and Σ are clear from the context.

Rewrite systems by themselves do not immediately lead to infinite-state systems. What is at least needed is a kind of *meta-rule* which prescribes *how* the rewrite rules have to be applied to words. Orthogonally to this concept, one can additionally introduce an equivalence relation on the set of words, which allows to model e.g. parallel processes by defining two words as equivalent if they are a permutation of each other. Formally, we will consider transition graphs generated by a rewrite system wrt. a meta-rule and an equivalence relation as follows.

Definition 2.3 (Labelled Rewrite Transition Systems)
Given a labelled rewrite system $\mathcal{R} = (V, \Sigma, R)$, a meta-rule

$$\vartheta : R \longrightarrow 2^{V^* \times \Sigma \times V^*}$$

and an equivalence relation θ on V^* the labelled rewrite transition system

$$\mathcal{T}(\mathcal{R}, \vartheta, \theta) =_{\mathrm{df}} (V^*/\theta, \Sigma, \to_{\mathcal{R},\vartheta,\theta})$$

is defined by the transition relation

$$\to_{\mathcal{R},\vartheta,\theta} =_{\mathrm{df}} \{\, [\gamma']_\theta \stackrel{a}{\to} [\gamma'']_\theta \;\mid\; \exists\, \alpha \stackrel{a}{\to} \beta \in R.\ (\gamma' \stackrel{a}{\to} \gamma'') \in \vartheta(\alpha \stackrel{a}{\to} \beta) \,\}$$

where $[\gamma]_\theta$ denotes the equivalence class of γ wrt. θ.

An example for a family of labelled rewrite transition systems is e.g. the class of BPA processes [BBK93] which is obtained by considering rewrite systems with *basic* rules of the form $A \stackrel{a}{\to} \alpha, \alpha \in V^*$, a meta-rule

$$\vartheta_P(\alpha \stackrel{a}{\to} \beta) \;\mapsto\; \{\, \alpha\gamma \stackrel{a}{\to} \beta\gamma \;\mid\; \gamma \in V^* \,\}$$

expressing *prefix rewriting*, and the identity relation on words. Taking instead an equivalence relation which identifies words up to permutation of letters yields the well-known class of BPP processes [Chr93].

In this paper we introduce a new *queue* meta-rule

$$\vartheta_Q(\alpha \stackrel{a}{\to} \beta) \;\mapsto\; \{\, \alpha\gamma \stackrel{a}{\to} \gamma\beta \;\mid\; \gamma \in V^* \,\}$$

which models in conjunction with the identity relation a first-in first-out discipline. This leads to processes different from the sequential and parallel case, as they exhibit a queue-like behaviour. An example of a rewrite system with basic rules, but interpreted wrt. the queue meta-rule is given in Figure 1.

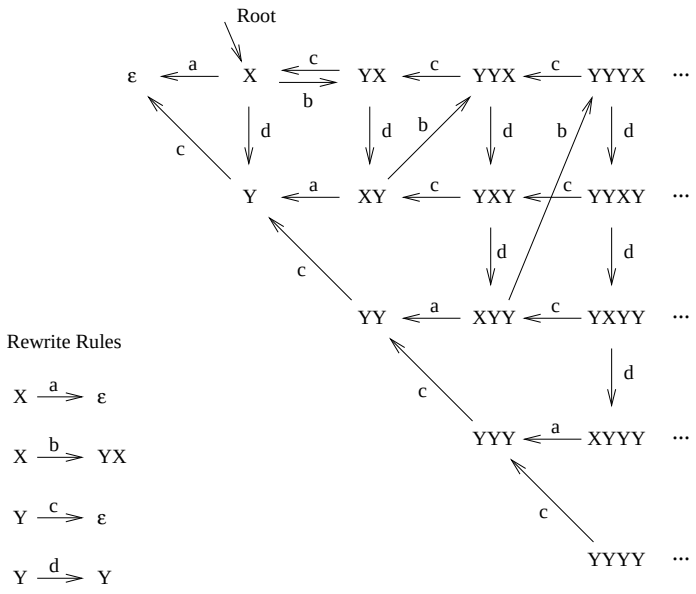


Figure 1: An example of a process with queuing discipline.

Similar to the taxonomy put forward in [Sti96, Mol96], restricting the form of allowed rewrite rules or adding a finite control yields different classes of processes. Here rewrite systems with a finite control have rewrite rules $(q', q'', \alpha \xrightarrow{a} \beta)$ and are applied to state-word pairs $q : \gamma$. As expected, a rule $(q', q'', \alpha \xrightarrow{a} \beta)$ rewrites $q_1 : \gamma_1$ to $q_2 : \gamma_2$ if $q_1 = q'$, $\alpha \xrightarrow{a} \beta$ rewrites γ_1 to γ_2, and $q_2 = q''$. Rewrite rules with finite control will henceforth be written as $q' : \alpha \xrightarrow{a} q'' : \beta$.

Classifying the families of transition systems according to the restrictions on the form of allowed rewrite rules, and on how words are interpreted we obtain the following hierarchies containing[1] Sequential Processes (SP), Parallel Processes (PP), Queue Processes (QP), their Basic (B) variants, as well as their Finitely Controlled (FC) versions.

Form of rewrite rules	Sequential	Parallel	Queue
$\alpha \xrightarrow{a} \beta$	SP = FCSP	PP (Petri Nets)	QP
$q'A \xrightarrow{a} q''\beta$	FCSP (PDA)	FCPP	FCQP
$A \xrightarrow{a} \beta$	BSP (BPA)	BPP	BQP

3 Reachability

In this section we investigate the reachability problem for the classes QP, FCQP, and BQP.

[1]The abbreviations used in the sequel are an attempt to standardize the naming conventions for these process classes.

First, it turns out that QP processes have already full Turing power and consequently an undecidable reachability problem. This result follows immediately from the undecidability of reachability for Post Tag systems with $k = 2$ [CM64].

Given an alphabet $V = \{ A_1, \ldots, A_n \}$, a *Post tag system* is a finite set of deterministic rewrite rules $\{ A_i \to \alpha_i, \alpha_i \in V^* \}$ together with a constant k. A computation step of a Post tag system transforms a word $a_1 \ldots a_n$ into $a_{k+1} \ldots a_n \alpha_j$ if $n \geq k$ and $a_1 = A_j$. This means that the first letter only determines the right-hand side of the rewrite rule which will be appended to the given word, after which the first k letters are removed from it.

Cocke and Minsky [CM64] have shown that Post tag systems with $k = 2$ are already universal, i.e. can simulate any Turing machine. We therefore have the following theorem.

Theorem 3.1 *Reachability for QP processes is undecidable.*

Proof: This result immediately follows from the observation that any Post tag system $\{ A_i \to \alpha_i, \alpha_i \in V^* \}$ over $V = \{ A_1, \ldots, A_n \}$ with $k = 2$ correponds to the queue process

$$\{ A_i A \to \alpha_i \mid A \in V, \alpha_i \in V^* \}$$

□

Adapting the idea of Cocke and Minsky we are able to prove universality also for FCQP processes which consequently have, as well as QP processes, an undecidable reachability problem. Of paramount importance in the proof is the ability to distinguish between sequences of odd and even length. In the QP case this is done by using two interleaved tapes, while in the case of FCQP we will have a state representing even, and one representing odd.

Theorem 3.2 *Reachability for FCQP processes is undecidable.*

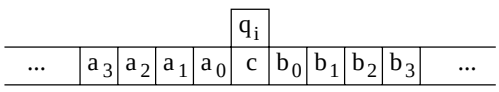
Proof: A Turing machine with a finite control, an input tape, and a head usually performs in the state q_i the following steps.

$$\text{State } q_i\text{:} \quad \text{Read } b_i\text{, if } b_i = \begin{cases} 0 & \text{then Write } B_{i0}, \text{Move } D_{i0}, \text{Goto } q_{i0} \\ 1 & \text{then Write } B_{i1}, \text{Move } D_{i1}, \text{Goto } q_{i1} \end{cases}$$

For the proof we will, however, use a variant of this machine model which was introduced in [CM64]. Here the reading of the tape is delayed and causes an immediate state-change.

$$\text{State } q_i\text{:} \quad \text{Write } B_i\text{, Move } D_i\text{, Read } b_i\text{, if } b_i = \begin{cases} 0 & \text{then Goto } q_{i0} \\ 1 & \text{then Goto } q_{i1} \end{cases}$$

Obviously, this model is equivalent to the standard model of a Turing machine. In our modified model a state of the Turing machine can now be represented by



As q_i represents the state *after* reading c, we do not need to include the letter c into our formal description. The sequence $\ldots a_3a_2a_1a_0$, respectively the sequence $\ldots b_3b_2b_1b_0$, will be interpreted as the integer $M =_{\mathrm{df}} \sum_{i=1}^{\infty} a_i 2^i$, respectively $N =_{\mathrm{df}} \sum_{i=1}^{\infty} b_i 2^i$. Since only a finite portion of the tape will ever contain non-blank symbols both integers are well defined. Consequently, a state of the Turing machine is fully described by the tuple (q_i, M, N).

A movement of the head can thus be modelled by manipulating M and N. For example, a move right means $M := 2M + B_i$, and $N := N/2$ if N is even, or $N := (N-1)/2$ if N is odd, while a move left just exchanges the roles of M and N in this transformation.

We are going to construct now a FCQP process faithfully simulating such a Turing machine. The finite control of the FCQP process to be constructed will consist of two states q_e (for even) and q_o (for odd). Suppose we have a configuration (q_i, M, N) which will be represented by the word $q_{e/o} : A_i a_i^M B_i b_i^N$. The state q_e (q_o) encodes that we have just read a zero (one), while the state q_i of the Turing machine is encoded in the tape contents, and not in the finite control. In the following we consider only a move right, as the case of moving left is dealt with dually, and omit transition labels as they are of no importance.
If we have to write in state q_i a 0 the FCQP process will contain the rules

$$q_{e/o} : A_i \to q_{e/o} : C_i, \quad q_{e/o} : a_i \to q_{e/o} : c_i c_i$$

whereas in case we have to write a 1 they will look as

$$q_{e/o} : A_i \to q_{e/o} : C_i c_i,\ q_{e/o} : a_i \to q_{e/o} : c_i c_i$$

Application of these rules yields a configuration $q_{e/o} : B_i b_i^N C_i c_i^{M'}$ where $M' = 2M$ if we have written a 0, and $M' = 2M + 1$ if we have written a 1. Applying the rewrite rules

$$q_{e/o} : B_i \to q_e : D_i, \qquad q_e : b_i \to q_o : \epsilon, \qquad q_o : b_i \to q_e : d_i$$

yields then the configuration

$$\begin{array}{ll} q_e : C_i c_i^{M'} D_i d_i^{N/2} & \text{if } N \text{ was even, i.e. we have read 0} \\ q_o : C_i c_i^{M'} D_i d_i^{(N-1)/2} & \text{if } N \text{ was odd, i.e. we have read 1} \end{array}$$

Now we finish the simulation of a single Turing machine cycle when N was even by applying the rewrite rules

$$\begin{array}{ll} q_e : C_i \to q_e : A_{i0}, & q_e : c_i \to q_e : a_{i0}, \\ q_e : D_i \to q_e : B_{i0}, & q_e : d_i \to q_e : b_{i0} \end{array}$$

producing the final configuration $q_e : A_{i0} a_{i0}^{M'} B_{i0} b_{i0}^{N/2}$ while in case N was odd we apply

$$\begin{array}{ll} q_o : C_i \to q_o : A_{i1}, & q_o : c_i \to q_o : a_{i1}, \\ q_o : D_i \to q_o : B_{i1}, & q_o : d_i \to q_o : b_{i1} \end{array}$$

yielding the final configuration $q_o : A_{i1}a_{i1}^{M'}B_{i1}b_{i1}^{(N-1)/2}$. The well-known undecidability of the halting problem for Turing machines implies, finally, that reachability for FCQP processes is undecidable. □

At this point we would like to observe that the simulation of a Turing machine by means of a FCQP process, as well as in the case of QP processes, is deterministic, which will play an important role when considering the associated bisimulation problems in the following section.

We close this section by mentioning that BQP processes correspond to Post tag systems with $k = 1$ for which Cook has settled the reachability problem in the affirmative. We therefore have the following theorem.

Theorem 3.3 ([Coo66]) *Reachability for BQP is decidable.*

4 Bisimulation

In this section we address the bisimulation problems for queue processes. Bisimulation equivalence plays a role of paramount importance in concurrency theory and is defined as follows [Par81, Mil89]:

Definition 4.1 *A binary relation R between processes is a* bisimulation *if whenever $(p, q) \in R$ then for each $a \in Act$:*

1. $p \xrightarrow{a} p'$ *implies* $\exists\, q'.\ q \xrightarrow{a} q' \ \wedge\ (p', q') \in R$, *and*
2. $q \xrightarrow{a} q'$ *implies* $\exists\, p'.\ p \xrightarrow{a} p' \ \wedge\ (p', q') \in R$.

Two processes p and q are said to be *bisimulation equivalent* or *bisimilar*, written $p \sim q$, if $(p, q) \in R$, for some bisimulation R.

Theorem 4.2 *Bisimulation is undecidable for QP and FCQP processes.*

Proof: As explained in Section 3 both classes of processes are universal. Given a Turing machine $\mathcal{M}$, we can construct a QP (FCQP) process $\mathcal{P}$ which faithfully and deterministically simulates $\mathcal{M}$. By using two copies $\mathcal{P}_1$ and $\mathcal{P}_2$ of $\mathcal{P}$ where the transition of the halting state is renamed to $halt_1$, respectively $halt_2$, we obtain

$$\mathcal{P}_1 \sim \mathcal{P}_2 \qquad \text{iff} \qquad \mathcal{M} \text{ does not halt}$$

□

In contrast, the bisimulation problem for BQP looks intriguing as BQP processes have not full Turing power, and are therefore not immediately excluded from possessing a decidable bisimulation problem. On the other hand the queue-like behaviour of these processes entails that concatenation is not a congruence wrt. $\sim$ as illustrated by the following example.

Example 4.3 *Let $A_1 \xrightarrow{a} BA_1, A_2 \xrightarrow{a} BA_1BA_1, B \xrightarrow{b} \epsilon$, and $C \xrightarrow{c} C$. Then we have $A_1 \sim A_2$, but $A_1C \not\sim A_2C$.*

Nevertheless, we conjecture that bisimulation is decidable for BQP processes.

5 Model Checking

In this section we are going to show that model checking the modal μ-calculus is undecidable for BQP processes. The well-known modal μ-calculus is a powerful branching time logic introduced by Kozen [Koz83]. It combines standard modal logic with least and greatest fixpoint operators which allows to express very complex temporal properties within this formalism. Formulas of the μ-calculus are defined by the following grammar

$$\Phi ::= \mathtt{tt} \mid X \mid \neg\Phi \mid \Phi \vee \Phi \mid \langle a \rangle \Phi \mid \mu X.\Phi$$

where X ranges over a (countable) set of variables Var, and a over a set of actions Σ. Additionally, we impose on the body of $\mu X.\Phi$ the syntactic restriction that any occurrences of X in Φ must occur within the scope of an even number of negations.

The semantics of μ-formulas is now given in Table 1. It is defined with respect to a labelled transition system $\mathcal{T} = (\mathcal{S}, \Sigma, \rightarrow)$, and a valuation $\mathcal{V}$ mapping variables to subsets of $\mathcal{S}$, where $\mathcal{V}[\, X \mapsto S \,]$ is the valuation obtained from $\mathcal{V}$ by updating the binding of X to S.

$$\begin{array}{rcl}
[\![\mathtt{tt}]\!]^{\mathcal{T}}_{\mathcal{V}} & =_{\mathrm{df}} & \mathcal{S} \\
[\![X]\!]^{\mathcal{T}}_{\mathcal{V}} & =_{\mathrm{df}} & \mathcal{V}(X) \\
[\![\neg\Phi]\!]^{\mathcal{T}}_{\mathcal{V}} & =_{\mathrm{df}} & \mathcal{S} \setminus [\![\Phi]\!]^{\mathcal{T}}_{\mathcal{V}} \\
[\![\Phi' \vee \Phi'']\!]^{\mathcal{T}}_{\mathcal{V}} & =_{\mathrm{df}} & [\![\Phi']\!]^{\mathcal{T}}_{\mathcal{V}} \cup [\![\Phi'']\!]^{\mathcal{T}}_{\mathcal{V}} \\
[\![\langle a \rangle \Phi]\!]^{\mathcal{T}}_{\mathcal{V}} & =_{\mathrm{df}} & \{\, s \in \mathcal{S} \mid \exists\, s' \in \mathcal{S}.\ s \xrightarrow{a} s' \text{ and } s' \in [\![\Phi]\!]^{\mathcal{T}}_{\mathcal{V}} \,\} \\
[\![\mu X.\Phi]\!]^{\mathcal{T}}_{\mathcal{V}} & =_{\mathrm{df}} & \bigcap \{\, S \subseteq \mathcal{S} \mid [\![\Phi]\!]^{\mathcal{T}}_{\mathcal{V}[\, X \mapsto S \,]} \subseteq S \,\}
\end{array}$$

Table 1: The semantics of μ-formulas.

The syntactic restriction imposed on the body of fixpoint operators ensures that the function which assigns $[\![\Phi]\!]^{\mathcal{T}}_{\mathcal{V}[\, X \mapsto S \,]}$ to a subset S is monotone. Hence, according to the Tarski-Knaster theorem [Tar55], it has a least fixpoint which gives the semantics of $\mu X.\Phi$.

Intuitively, the semantics express that all states satisfy $\mathtt{tt}$, s satisfies X if s is an element of the set bound to X in the current valuation, s satisfies $\neg\Phi$ if it does not satisfy Φ, and s satisfies $\Phi_1 \vee \Phi_2$ if it satisfies Φ_1 or Φ_2. The modal operator $\langle a \rangle$ then admits to specify the existence of transitions, as s satisfies $\langle a \rangle \Phi$ if there exists s' reachable via an a-transition from s, and satisfying Φ. Finally, the fixpoint operator μ allows to specify some infinite behaviour, since $\mu X.\Phi$ denotes the least fixpoint of the functional Φ with input parameter X over the powerset of $\mathcal{S}$.

We prove now the undecidability of the model checking problem for BQP and the modal μ-calculus by means of a reduction from the halting problem of two-counter machines.

A two-counter machine $\mathcal{M}$ has a set of states $\{ q_0, \ldots, q_{n+1} \}$, two counters $\{ c_1, c_2 \}$ and a set of transition rules $\{ \delta_0, \ldots, \delta_n \}$. A transition rule δ_k describes the action to be taken when the machine is in state q_k and is either of the form

(I) $\quad q_k : c_i := c_i + 1;$ goto q_l

or of the form

(II) $\quad q_k :$ if $c_i = 0$ then goto q_{l_1} else $\{c_i := c_i - 1;$ goto $q_{l_2}\}$

A configuration of $\mathcal{M}$ is a tuple (q_k, n_1, n_2) where n_1, n_2 are integers representing the contents of the counters c_1, c_2. In particular, the initial configuration is $(q_0, 0, 0)$. A computation of $\mathcal{M}$ is then a sequence of configurations beginning with the initial one and proceeding by applying succesively the transition rules to the current configuration. Observe that computations of two-counter machines are always deterministic, as each state has at most one transition rule. A machine $\mathcal{M}$ halts if its computation is finite, i.e. reaches the state q_{n+1}. Minsky has shown that two-counter machines are Turing equivalent, and consequently have an undecidable halting problem [Min67].

Theorem 5.1 *The model checking problem for the class of BQP processes and the modal μ-calculus is undecidable.*

Proof: Given a two-counter machine $\mathcal{M}$, we construct a BQP process $\mathrm{BQP}(\mathcal{M})$ as follows.

$$
\begin{array}{ll}
q_k \stackrel{\mathrm{inc}_i}{\rightarrow} q_l c_i & \text{if } \delta_k \text{ is of the form (I)} \\
\left.\begin{array}{l} q_k \stackrel{\mathrm{zero}_i}{\rightarrow} q_{l_1} \\ q_k \stackrel{\mathrm{dec}_i^1}{\rightarrow} q_{l_2}, c_i \stackrel{\mathrm{dec}_i^2}{\rightarrow} \epsilon \end{array}\right\} & \text{if } \delta_k \text{ is of the form (II)} \\
q_{n+1} \stackrel{\mathrm{halt}}{\rightarrow} \epsilon & \\
\left.\begin{array}{l} c_i \stackrel{\mathrm{ex}_i}{\rightarrow} c_i, \quad c_i \stackrel{\mathrm{shift}}{\rightarrow} c_i \end{array}\right\} & \text{for } i = 1, 2
\end{array}
$$

Notice that the BQP process constructed models the given two-counter machine only in a *weak* sense: there exists an "honest" transition sequence beginning from q_0 that represents the computation of $\mathcal{M}$, but the remaining sequences are "dishonest", as they may decrease a counter by more than one, or may take the zero branch although the counter is not zero at all.

Using the modal μ-calculus it is, however, possible to specify a formula *Halt* expressing that a transition sequence is finite, honest and ends with the halt action.

$$
\begin{array}{lcll}
\mathit{Shift}(Z) & \equiv & \mu Y. & \langle \mathrm{shift} \rangle Y \vee Z \\
\mathit{Halt} & \equiv & \mu X. & \bigvee_{i=1,2} \langle \mathrm{inc}_i \rangle \ \mathit{Shift}(X) \\
 & & & \bigvee_{i=1,2} \langle \mathrm{zero}_i \rangle \ (\mathit{Shift}(X) \wedge \neg \mathit{Shift}(\langle \mathrm{ex}_i \rangle \mathtt{tt})) \\
 & & & \bigvee_{i=1,2} \langle \mathrm{dec}_i^1 \rangle \ \mathit{Shift}(\langle \mathrm{dec}_i^2 \rangle \mathit{Shift}(X)) \\
 & & & \vee \langle \mathrm{halt} \rangle \ \mathtt{tt}
\end{array}
$$

Since we have now

$$\mathrm{BQP}(\mathcal{M}) \text{ satisfies } \mathit{Halt} \quad \text{iff} \quad \mathcal{M} \text{ halts}$$

this proves the undecidability of model checking the modal μ-calculus for BQP. □

6 Conclusions and Further Work

In this paper we have introduced a new hierarchy of processes with a queue-like behaviour which fits naturally into the taxonomy of process classes defined by rewrite systems. From a practical point of view they extend the spectrum of formalisms for the description of processes by emphasizing the model of a queue, which has to be simulated in other frameworks.

While we have proved the (un)decidability of some reachability, bisimulation and model-checking problems for queue processes these results can only be seen as a first step. Still open interesting questions are the bisimulation problem for BQP, deciding finiteness questions wrt. bisimulation equivalence, as well as model checking weaker branching time logics, and linear time logics, like e.g. LTL. Moreover, the exact relationship with the hierarchy of sequential and parallel processes seems also to be of theoretical interest.

References

[BBK93] J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. Decidability of Bisimulation Equivalence for Processes Generating Context-Free Languages. *Journal of the ACM*, 40(3):653–682, 1993.

[BE97] O. Burkart and J. Esparza. More Infinite Results. In *INFINITY '96*, volume 6 of *ENTCS*, page 23. Elsevier Science B.V., 1997.

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

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

[CM64] J. Cocke and M. Minsky. Universality of Tag Systems with P=2. *Journal of the ACM*, 11(1):15–20, 1964.

[Coo66] S.A. Cook. The Solvability of the Derivability Problem for One-Normal Systems. *Journal of the ACM*, 13(2):223–225, 1966.

[Esp97] J. Esparza. Decidability of Model-Checking for Concurrent Infinite-State Systems. *Acta Informatica*, 34:85–107, 1997.

[Koz83] D. Kozen. Results on the Propositional μ-Calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice–Hall, 1989.

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

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

[Par81] D. Park. Concurrency and Automata on Infinite Sequences. In *5th GI Conference*, LNCS 104, pages 167–183. Springer, 1981.

[Sti96] C. Stirling. Decidability of Bisimilarity. In *INFINITY '96*, Technical Report MIP-9614, pages 30–31. University of Passau, July 1996.

[Tar55] A. Tarski. A Lattice-Theoretical Fixpoint Theorem and its Applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

Faithful π-nets.
A graphical representation of the π-calculus.

Gabriel Ciobanu and Mihai Rotaru
A.I.Cuza University
Dept. of Computer Science
6600 Iasi, Romania
gabriel@info.uaic.ro mrotaru@iit.tuiasi.ro

Abstract

Starting from the π-nets defined by Robin Milner, we present a graphical formalism called *faithful π-nets*. The aim of these faithful π-nets is to describe graphically the systems with a dynamical changing configurations. The π-nets[Mil94] give an "approximate" representation for the π-calculus; the correspondence between the π-nets and the π-calculus is not very accurate. The "faithful π-nets" correspond precisely to the π-calculus constructions. Moreover, they are simple and intuitive. The paper gives two "fully abstract" translations **draw** and **write** from the π-calculus to the faithful π-nets, and back. The congruence used for these full-abstraction results is the barbed bisimulation. Operational correspondence results are given for both translation. For every π-net G without isolated nodes, and for every π-term P we have **draw**(**write** (G)) $= G$ and **write**(**draw** (P)) $\equiv P$. The paper describes also a simple graphical encoding for lazy λ-calculus by faithful π-nets, avoiding the Honda-Tokoro[HT91] and Boudol[Bou92] transformations - which could lead also to graphical encodings of the lazy λ-calculus.

1 Introduction

This paper presents the *faithful π-nets*, a graphical formalism which is equivalent to the π-calculus, and which provides a good and simple graphical representation of the π-calculus. There are some other attempts to give a graphical representation for the π-calculus: π-nets [Mil94], interaction diagrams[Par93], Yoshida's "graph notation" [Yos94], and graph rewriting systems [MP95]. Parrow introduced the interaction diagrams to describe graphically the constructions of the π-calculus. Intuitively, these diagrams correspond closely to the π-calculus terms; however, as far as the authors of this paper know, there are no results expressing precisely the relationship between interaction diagrams and π-calculus. On the other hand, our faithful π-nets look simpler than the interaction diagrams. Yoshida

gave a very accurate encoding of the π-calculus syntactical terms, correct in all details; her encoding is given in terms of the concurrent combinators of the π-calculus, and in this way it doesn't hold, or at least it is not clear if it holds a "faithfulness" property. In [MP95] the authors give an operational semantics for the π-calculus mapping its language into a graph rewriting system. The graph rewriting systems describe the evolution of the π-terms, and there is no translation (mapping) from graphs to π-terms.

The root of our approach is defined by the π-nets introduced by Robin Milner in [Mil94] as a graphical action calculus [Mil93a]. The notion of action calculus was introduced in an attempt to bring some uniformity into the study of behavioural calculi such as the λ-calculus, the π-calculus, Petri nets, ... The actions of an action calculus can be drawn as a special kind of graphs. Thus reductions in the λ-calculus, and the π-calculus reactions are in fact special cases of graph reductions over these graphs (and this is a starting key point of our approach). The π-nets are not presented as a graphical version of the π-calculus, but as an action calculus corresponding to the π-calculus. The paper [Mil94] described in an informal way how the π-calculus is embedded into a formalism defined by the π-nets. An accurate and direct mapping from a π-term to a π-net is somehow difficult, and a reason is that actually the π-nets are actions. In this way each π-net a has an arity $(a : m \to n)$, a fact which has no correspondence in the π-calculus. The π-terms are analogous to the π-nets with arity $0 \to 0$. The π-nets are more interesting if they are considered from the viewpoint of the action calculus, and perhaps they are more general than the π-calculus, mainly because of their algebraic structure. However it is important to note that the graphical presentation of the π-calculus does not use the algebraic structure of the π-nets too much; only π-nets of arity $0 \to 0$ are used. Starting from this remark we study the subset of the π-nets of arity $0 \to 0$ as an independent formalism, and we give a rather simple graphical representation for the π-calculus.

In this paper we show that the faithful π-nets and the π-calculus have the same expressive power, and we define two "fully abstract" translations - from π-calculus to faithful π-nets, and back. These encodings are "fully abstract" when two source calculus terms are equivalent if and only if their translations are equivalent. The congruence used for these two full-abstraction results is the barbed bisimulation, which can be defined uniformly in both calculi - and in many other process calculi as well. In order to show how these results are actually related to the operational semantics, a connection between reductions over terms and reductions over their encodings is given by two results revealing the operational correspondence for both translations.

The paper is organized as follows. Section 2 reviews the basic definitions of π-calculus. The graphical formalism given by the faithful π-nets is presented in Section 3. Section 4 shows that the formalisms represented by the π-calculus and the faithful π-nets are equivalent; some examples describe how the faithful π-nets work. Section 5 gives a graphical encoding of the lazy λ-calculus by the faithful π-nets. The concluding section briefly presents some remarks and a possible direction for future work.

The paper is self-contained; however, knowledge of the π-calculus, the π-nets, the action structures and action calculi should help the understanding.

2 π-calculus

First we introduce the formal π-calculus framework. We consider the monadic π-calculus without output guards [Mil91]. It is known that monadic π-calculus with only input guards has the full power of polyadic π-calculus; therefore this restriction doesn't affect the expressiveness, because the output guarding can be defined in terms of input guarding [HT91, Bou92]. Therefore we don't use the output guards $\overline{x}\langle z\rangle.P$, but only the *output messages* $\overline{x}\langle z\rangle$ to denote the emission of a name z along a channel x. (In this way we have an asynchronous version of the π-calculus.) Let $\mathbf{N}$ be a countable set of *names*. The elements of $\mathbf{N}$ are denoted by $x, y \ldots$. The terms of this formalism are called *processes*. The set of processes is denoted by $\mathbf{P}$, and processes are denoted by $P, Q, R \ldots$.

Definition 1 *The processes are defined over the set* $\mathbf{N}$ *of names by the following syntactical rules:*

$$P \ ::= \ 0 \quad | \quad \overline{x}\langle z\rangle \quad | \quad x(y).P \quad | \quad !x(y).P \quad | \quad (\nu x)P \quad | \quad (P \mid Q) \tag{1}$$

The prefix $x(y)$ binds the name y, and (νx) binds the name x. We denote by $fn(P)$ the set of the names with free occurrences in P. We denote by $P\{v/u\}$ the result of simultaneous substitution in P of all free occurrences of the name u by the name v, using the α-conversion wherever necessary to avoid the name capture. An *input guard* $x(y).P$ denotes the reception of an arbitrary name z along channel x, and afterwards behaving as $P\{z/y\}$. A *replicated input guard* $!x(y).P$ denotes a process that allows to generate arbitrary instances of the form $P\{z/y\}$ in parallel by repeatedly receiving names z along channel x. The informal meaning of restriction $(\nu x)P$ and parallel composition $P \mid Q$ is as usual.

Over the set of processes it is defined a *structural congruence* relation; this relation defines a syntactical equivalence over processes, providing somehow a non-explicit semantics of some formal constructions. Milner imposed this structural congruence upon the π-calculus as part of the formal language, and not as part of its semantics[Mil92].

Definition 2 *The relation* $\equiv \subseteq \mathbf{P} \times \mathbf{P}$ *is called structural congruence, and it is defined as the smallest congruence over processes which satisfies the following requirements:*

1. $P \equiv Q$, *if* P *is* α*-convertible to* Q*;*
2. $P \mid 0 \equiv P$, $P \mid Q \equiv Q \mid P$, $(P \mid Q) \mid R \equiv P \mid (Q \mid R)$*;*
3. $(\nu x)0 \equiv 0, \quad (\nu x)(\nu y)P \equiv (\nu y)(\nu x)P, \quad$ *and*
   $(\nu x)(P \mid Q) \equiv (\nu x)P \mid Q, \quad$ *if* $x \notin fn(Q)$.

The structural congruence deals with the aspects related to the structure of the processes, not to their mobility. In this way the structural aspects will not appear in the rules of the reaction relation which deals mainly with mobility and interaction.

Definition 3 *The reaction relation over processes is defined as the smallest relation* $\rightarrow \subseteq P \times P$ *satisfying the following rules:*

$$(COM) \qquad \overline{x}\langle z\rangle \mid x(y).P \rightarrow P\{z/y\}$$

$$(REP) \qquad \overline{x}\langle z\rangle \mid !x(y).P \rightarrow P\{z/y\} \mid !x(y).P$$

$$(PAR)\frac{P \rightarrow P'}{P \mid Q \rightarrow P' \mid Q} \qquad (RES)\frac{P \rightarrow P'}{(\nu x)P \rightarrow (\nu x)P'}$$

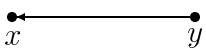
$$(STRUCT)\frac{P \equiv P' \quad P' \rightarrow Q' \quad Q' \equiv Q}{P \rightarrow Q}$$

3 The faithful π-nets

We define the faithful π-nets as a process algebra, by considering some ground faithful π-nets, and then composing them by some operators. The ground graphical representations and the constructors (operators) of the faithful π-net algebra correspond to the syntactic constructors of the π-calculus. We use similar notations. As a consequence, we obtain a "textual" representation for our graphical constructions. This one-dimensional syntactical representation provides an easier formal way of expressing the properties of the two-dimensional graphical formalism. However we have in mind, and essentially we discuss about the graphical representations.
Roughly speaking, a faithful π-net is a tree with graphs as nodes, together with an injective map which assigns labels to the nodes of these graphs. Formally, let X be the set of names from π-calculus, and let $\#$ be an extra special symbol. We denote by $x, y, \ldots$ the elements of X, and by $u, v, \ldots$ the elements of $X \cup \{\#\}$. Therefore a label of a node could be a name or $\#$; the meaning of $\#$ is that it doesn't matter what is the label of the corresponding node - and this is the reason why we don't give it a proper label.
We denote by $G, H, \ldots$ the π-nets, and by l(G) the set of labels of a π-net G. The formal definitions for faithful π-nets, labels, as well as the representation of nets as expressions are given in the following definition.

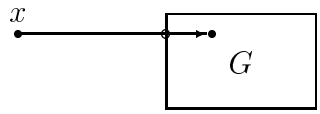
Definition 4 *The set* Π *of the faithful* π*-nets is defined inductively by:*

- *[atomic]* $0_u \in \Pi$*, with* $l(0_u) = \{u\} \setminus \{\#\}$*, is the* π*-net with a single node which is labelled by a name* u*;*
- *[message]* $\leftarrow xy \in \Pi$*, with* $l(\leftarrow xy) = \{x, y\}$*, is the* π*-net*



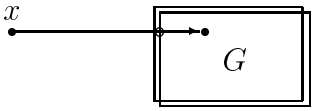
This π*-net corresponds to the output* $\overline{x}\langle z\rangle$*.*

- *[boxing] If $a \in \Pi$, then*
  - *i) $x.y[G] \in \Pi$, with $l(x.y[G]) = \{x\} \cup (l(G) \setminus \{y\})$, is the π-net*



This π-net corresponds to the input guard $x(y).P$.

  - *ii) $x : y[G] \in \Pi$, with $l(x : y[G]) = \{x\} \cup (l(G) \setminus \{y\})$, is the π-net*



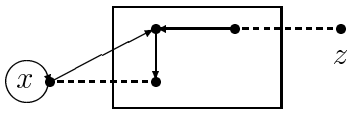
This π-net corresponds to the replicated input guard $!x(y).P$.

The arrows of the last two π-nets come into the node of G which is labelled by y, and we replace the label y by # . This means that we prefer to keep this node unlabelled, and we understand from now on that # is assigned to each unlabelled node of G. When we overlap a node labelled by x and a node labelled by #, the resulting node is labelled by x. On the other hand, every properly labelled node which is inside of a box is "exported" outside the box, and it is linked to the box by a pseudo-arrow, called connection, in this way: 

Connection is a construction of our formal approach.

- *[juxtaposition] $G \otimes H \in \Pi$, with $l(G \otimes H) = l(G) \cup l(H)$, is the faithful π-net obtained by fitting G and H, namely identifying the nodes of G with the nodes of H having the same proper label;*
- *[relabelling] $G[u/x] \in \Pi$, with $l(G[u/x]) = (l(G) \setminus \{x\}) \cup (\{u\} \setminus \{\#\})$, is the faithful π-net obtained from G by substitution of the label x by the label u, eventually identifying the nodes with the same name label.*

Example 1 *The π-net described by $G = \leftarrow xx \otimes x.y[\leftarrow xy \otimes \leftarrow yz]$ is represented by*



Definition 5 *A node which is the source or the target of a proper arrow (no connection) is called non-isolated, or non-singular. The nodes linked by a connection to a non-singular node are also non-singular. The other nodes are called isolated, or singular.*

We define a reduction relation over these graphical constructions by using their textual representations.

Definition 6 *The reduction relation is the smallest relation over* Π *generated by the following rules:*

(com) $\leftarrow xz \otimes x.y[G] \rightarrow G[z/y]$,
(rep) $\leftarrow xz \otimes x : y[G] \rightarrow G[z/y] \otimes x : y[G]$,
(par) *if* $G \rightarrow G'$ *then* $G \otimes H \rightarrow G' \otimes H$,
(res) *if* $G \rightarrow G'$ *then* $G[\#/x] \rightarrow G'[\#/x]$.

Remark 1

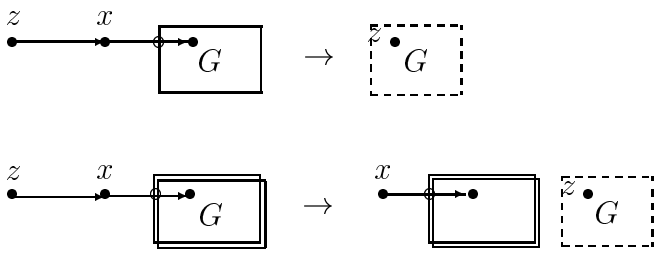
- *i) The rule* (com) *corresponds to the rule* (COM) *of definition 3, and the rule* (rep) *corresponds to the rule* (REP) *of definition 3.*
- *ii) We may skip the rule* (rep) *if we introduce an equivalence relation* $\cong$ *over the faithful* π*-nets which is compatible with the rules for* $\otimes$ *and* $[\#/x]$ *such that*
$$u : x[G] \cong u.x[G \otimes u : x[G]] \ .$$

We remove the isolated nodes of a π-net by using a deleting rule similar to the cooling rule of CHAM [BB92]. Thus we have the following definitions:

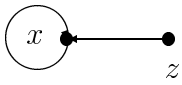
Definition 7 *The relation* $\rightharpoonup_1$ *is the smallest relation over the* π*-nets satisfying the rule:*
$$0_u \otimes G \rightharpoonup_1 G \quad \textit{if} \quad u \notin l(G) \ .$$

Definition 8 $\rightharpoonup = (\rightharpoonup_1)^+$

We can give now more details on the graphical representations for the rules which define the reduction relation. For instance, the rule (com) makes known that after a reduction two arrows are deleted and two nodes (the source of the first arrow, and the target of the second one) are treated as identical. If the linking middle node x becomes isolated, then it is deleted by $\rightharpoonup$. The box is deleted, the source and target nodes of the connections corresponding to this box are overlapping, and we suggest this by using a dash box. These dash boxes are not constructions of the formalism we are defining.
The rules (com) and (rep) can be represented by the following reaction relation over the faithful π-nets:
(com) $\leftarrow xz \otimes x.y[G] \rightarrow G[z/y]$,
describes $\overline{x}\langle z\rangle \mid x(y).P \rightarrow P\{z/y\}$, and
(rep) $\leftarrow xz \otimes x : y[G] \rightarrow G[z/y] \otimes x : y[G]$,
describes $\overline{x}\langle z\rangle \mid !x(y).P \rightarrow P\{z/y\} \mid !x(y).P$



Example 1.(continuation) Using these rules over the faithful π-nets, the π-net described by the *Example 1.* can be reduced to



4 The π-calculus and the faithful π-nets are equivalent

Let $\mathcal{P}_1$, $\mathcal{P}_2$ be two process calculi, and $\sim_1$, $\sim_2$ two corresponding equivalences over them.

Definition 9 *i) $\mathcal{P}_2$ is more expressive than $\mathcal{P}_1$ if there is a full abstract translation T : $\mathcal{P}_1 \to \mathcal{P}_2$, i.e. for every $P, Q \in \mathcal{P}_1$,*

$$P \sim_1 Q \qquad \textit{iff} \qquad T(P) \sim_2 T(Q)$$

ii) $\mathcal{P}_1$ and $\mathcal{P}_2$ have the same expressive power if $\mathcal{P}_2$ is more expressive than $\mathcal{P}_1$ and $\mathcal{P}_1$ is more expressive than $\mathcal{P}_2$.

Following Park and Milner, the most studied forms of behavioural equivalence in process algebras are based on the notion of bisimulation. The standard definition of bisimulation (i.e., the one for CCS and related calculi), as well as higher-order bisimulation are in general unsatisfactory (particularly in higher-order calculi) because over-discriminating. We use in this paper the Milner&Sangiorgi's barbed bisimulation. One of its advantage is that it can be defined uniformly in different calculi (including higher-order ones). Previous experimentations with CCS and π-calculus have shown that its congruence, called barbed congruence, yields a desired discriminanting power. It was proved that barbed congruence also gives a natural bisimilarity congruence in higher-order calculi.
In order to prove that π-calculus and faithful π-nets have the same expressive power, we use the *observational congruence* as the equivalences $\sim_1$, $\sim_2$ of the previous definition. The observational congruence is based on "barbed bisimulation", a bisimulation which can be defined uniformly in many process calculi including those we consider in this paper. Moreover, we don't use a labelled transition system; we focus on a reduction relation, and we use some observation predicates (which can be easily defined for our π-nets).

Definition 10

i) We consider the following predicates over processes:

$P \downarrow_y$= *true if* y *if there is a prefix* $y(x)$ *or* $\overline{y}x$ *which is not underneath another prefix and not in the scope of a restriction* (νy)*;*

ii) We consider the following predicates over π*-nets:*

$G \downarrow_y$= *true if* y *is the label of a non-isolated node.*

Each predicate $\downarrow_y$= *true* detect the possibility of performing an interaction with the environment along y.

Definition 11 *Strong barbed bisimulation, written* $\sim'$*, is defined over the processes of* π*-calculus, as well as over the* π*-nets, as the largest symmetrical relation such that* $T \sim' S$ *implies:*

i) whenever $T \rightarrow T'$ *then there exists* S' *s.t.* $S \rightarrow S'$ *and* $T' \sim' S'$*;*

ii) for each u*, if* $T \downarrow_u$ *then* $S \downarrow_u$*.*

Definition 12 *Two terms* T *and* S *are strong barbed congruent, and we write* $T \sim S$*, if for each (textual) context* $C[\cdot]$*, it holds that* $C[T] \sim C[S]$*.*

Some useful remarks on barbed congruences can be found in [San93]. Davide Sangiorgi describes an encoding of the term-passing π-calculus into the ordinary π-calculus in this nice paper.

4.1 From π-calculus to faithful π-nets

We start to show that the π-calculus and the faithful π-nets have the same expressive power by describing how the processes of the π-calculus are translated into faithful π-nets.

Definition 13 *The graphical representation of the* π*-calculus by the faithful* π*-nets is given by the function*

$\mathbf{draw} : \mathbf{P}/\equiv \rightarrow \Pi$

which is defined by:

- $\mathbf{draw}(0) = 0_{\#}$
- $\mathbf{draw}(\overline{x}\langle z\rangle) = \leftarrow xz$
- $\mathbf{draw}(x(y).P) = x.y[\mathbf{draw}(P) \otimes 0_y]$
- $\mathbf{draw}(!x(y).P) = x : y[\mathbf{draw}(P) \otimes 0_y]$
- $\mathbf{draw}((\nu x)P) = (\mathbf{draw}(P) \otimes 0_x)[\#/x]$
- $\mathbf{draw}(P \mid Q) = \mathbf{draw}(P) \otimes \mathbf{draw}(Q)$

Regarding to this translation we have the following results:

Lemma 1 $fn(P) = l(\mathbf{draw}(P))$.

Lemma 2 draw *is well-defined, i.e. it doesn't depend on the choice of a process (the representative) of an equivalence class of processes:*

$$P \equiv Q \quad \textit{implies} \quad \mathbf{draw}(P) = \mathbf{draw}(Q).$$

The following result shows that if two processes have the same behaviour (are strong barbed congruent), then their corresponding π-nets have the same behaviour (are strong barbed congruent).

Proposition 1 ((full abstraction for draw))

$$P \sim Q \qquad \textit{iff} \qquad \mathbf{draw}(P) \sim \mathbf{draw}(Q)$$

Following a remark of Davide Sangiorgi [San93], nothing prevents us from obtaining the same result with a very bizarre encoding. This means that we should reveal the operational correspondence existing between P and $\mathbf{draw}(P)$. The next result shows this correspondence, and that the translation by **draw** is really interesting and faithful.

Proposition 2 ((operational correspondence for draw))

- *If* $P \to P'$ *then* $\mathbf{draw}(P) \to\!\!\rightharpoonup \mathbf{draw}(P')$*;*
- *If* $\mathbf{draw}(P) \to\!\!\rightharpoonup G \not\rightarrow_1$*, then there exist* P' *such that*
  $P \to P'$ *and* $G = \mathbf{draw}(P')$

4.2 From faithful π-nets to π-calculus

The converse of the previous translation is given by an algorithm.

Let G be a π-net. We assign new symbolic labels to all *unlabelled* nodes (in fact these nodes are labelled by #) by the following procedure - where we take in consideration the order of its steps:

- First, we assign labels $(x_i),\ i \geq 0$
  - to those nodes which are connected, but they are not the targets of any arrow ("external" nodes),
  - to those nodes which are the targets of $\stackrel{\circ}{\longrightarrow}$ arrows (i.e arrows obtained by applying the "boxing" rules of the definition 4),
  - to every (connected) node which is still unlabelled, applying the rule that two connected nodes receive the same label;
- Second, we assign labels $(y_j),\ j \geq 0$ to the other nodes which are still unlabelled.

When we apply the algorithm, these symbolic labels are working effectively only for those subnets of G which are still valid π-nets.

function **write**(var G : π-net) : π-term

1. **write** = 0
2. *while* (there is at least a box in G) *do*
   - select a box which is not included into another box; let H be this box, and let $y \stackrel{\circ}{\longrightarrow} x$ its corresponding arrow (see the [boxing] rules of the definition 4);
   - **write** = (**write** | (!)y(x).**write**(H));
   - delete the chosen box;
3. *while* (there is at least an arrow in G) *do*
   - select an arrow ($x \rightarrow y$);
   - **write** = (**write** | $\overline{y}x$);
   - delete the chosen arrow;
4. **write** = $(\nu\tilde{x})(\nu\tilde{y})$ **write**.

where $\tilde{x}$ refers to a label x of some "external" nodes of G, and $\tilde{y}$ refers to a label y which is one of the symbolic labels (y_i) of the previous procedure.

Regarding to this translation from our faithful π-nets to the π-calculus processes, we have the following results:

Lemma 3 *If G has no isolated nodes, then $l(G) = fn(\mathbf{write}(G))$.*

This algorithm selects a box, and then selects an arrow. It is clear that we might obtain different encodings for the same faithful π-net. Fortunately we have the following result:

Lemma 4 *If P and Q are two different π-terms obtained by applying the algorithm* **write** *over a π-net G, then $P \equiv Q$.*

Proposition 3 ((full abstraction for write))

$$G \sim H \qquad \textit{iff} \qquad write(G) \sim write(H)$$

Proposition 4 ((operational correspondence for write))

- *If $G \rightarrow G'$ then* $\mathbf{write}(G) \rightarrow \mathbf{write}(G')$*;*
- *If* $\mathbf{write}(G) \rightarrow P$ *then there exist G' such that $G \rightarrow G'$ and $P \equiv \mathbf{write}(G')$.*

Moreover,

Proposition 5 *For every π-net G without isolated nodes, and for every π-term P we have the following results:*

- $\mathbf{draw}(\mathbf{write}\ (G)) = G$*;*
- $\mathbf{write}(\mathbf{draw}\ (P)) \equiv P$.

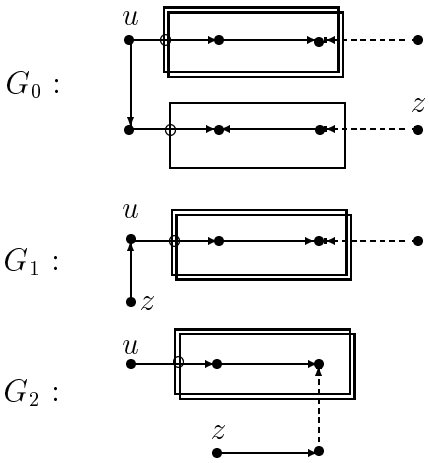
4.3 Examples (of how the faithful π-nets work).

We give here some examples which show the dynamics of the faithful π-nets, the way how the reductions work over the graphical representations of the π-terms.

Example 2 *We consider the following π-term:*

$$P = (\nu x)(x(y).\overline{y}\langle z\rangle.0 \mid (\nu w)(\overline{x}\langle u\rangle.0 \mid (\nu t)(!u(r).t\langle r\rangle))) \tag{2}$$

*The corresponding π-net $G_0 =$**draw**(P), and its reactions ($G_0 \to G_1 \to G_2 \not\to$) are described by*



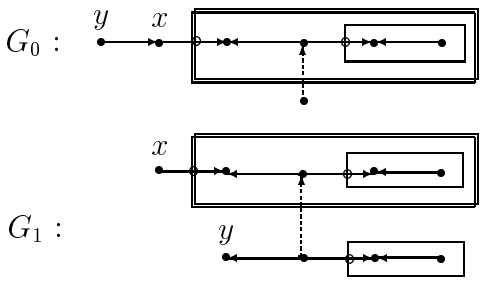
Example 3 *We consider now the process*

$$P = (\nu t)(\overline{x}\langle y\rangle \mid !x(u).(\overline{u}\langle t\rangle \mid t(s).(\nu r)\overline{s}\langle r\rangle))$$

Applying the reaction relation $P \to P'$, we obtain the following process P'

$$P' = (\nu t)(!x(u).(\overline{u}\langle t\rangle \mid t(s).(\nu r)\overline{s}\langle r\rangle) \mid (\overline{y}\langle t\rangle \mid t(s).(\nu r)\overline{s}\langle r\rangle))$$

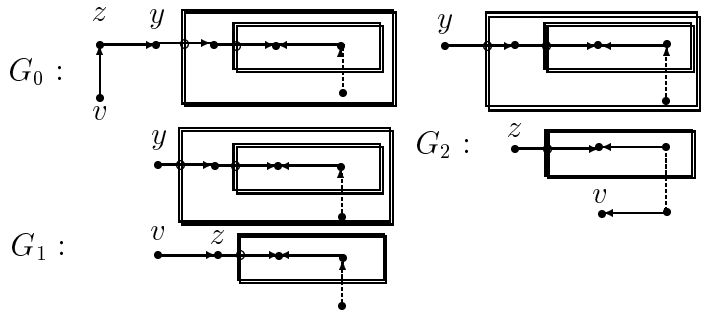
*The similar reaction for the faithful π-nets is given by $G_0 =$ **draw**$(P) \to G_1 =$ **draw**(P'), where*



Example 4 *This example shows how the reactions over the faithful π-nets are similar to those over the processes of the π-calculus. We consider the process*

$$P = (\overline{z}\langle v\rangle \mid \overline{y}\langle z\rangle \mid !y(x).((\nu t)!x(u).\overline{u}\langle t\rangle))$$

Starting from the faithful π-net G_0 which corresponds to the process P, two reactions ($G_0 \rightarrow G_1 \rightarrow G_2$) are described by the following picture



5 A graphical presentation of the λ-calculus by the faithful π-nets

The paper [Mil92] is an important paper on mobile processes, and it is invaluable for a good exposition of what is going on in the interaction between mobile processes, and also for studying how one may go from concurrent processes to functions. A deep investigation into Robin Milner's encoding of lambda-calculus introduced in [Mil92] is given by Davide Sangiorgi in [San94].
There are at least two reasons why the translation of the λ-calculus into various formal models for concurrency and interaction is interesting: first reason is concerning the expressive power of the model, and the second is related to the new properties of the λ-calculus which could be obtained as a consequence of such a translation into a more general context (from the point of view of concurrency and interaction) - see [Mil92, San92]. We present now a translation which embeds the λ-calculus into a *graphical concurrent context.*

A graphical translation could be done by starting from the translation of the lazy λ-calculus into π-calculus described by Robin Milner in [Mil92], and using then the translation described by Honda and Tokoro in [HT91]. In this way the λ-terms are translated into π-terms, and these π-terms are translated by the Honda-Tokoro construction into π-terms without output guards which have a graphical representation by π-nets. We present a rather simple and direct translation using the faithful π-nets. For this encoding we takes into consideration the translation of the lazy λ-calculus into the π-calculus, but we avoid the Honda-Tokoro[HT91] and Boudol[Bou92] transformations which could lead also to graphical encodings.

5.1 Lazy λ-calculus

We recall the basic notions of the lazy version of λ-calculus, according to [Abr89]. We consider a set $\mathbf{V}$ of variables, and we denote its elements by $x, y, \ldots$.

Definition 14 *The λ-terms are defined over the set $\mathbf{V}$ of variables by the following syntactical rules:*

$$a ::= x \quad | \quad \lambda x.a \quad | \quad ab$$

Considering the λ-term $\lambda x.\, a$, every occurrence of x into a is bound. The λ-terms are denoted by $a, b, \ldots$, and the set of the λ-terms is denoted by L.

Definition 15 *The reaction relation over λ-terms is defined as the smallest relation $\rightarrow \subseteq L \times L$ satisfying the following rules:*

$$(\beta) \qquad (\lambda x.a)b \rightarrow a\{b/x\}$$

$$(appl) \qquad \frac{a \rightarrow a'}{ab \rightarrow a'b}$$

This relation defines the so called "lazy" version of the λ-calculus. α-conversion is allowed. $a\{b/x\}$ represents the λ-term a in which the free occurrences of x are substituted by b (avoiding the name capture).

5.2 Translation

We consider $\mathbf{V} \subset \mathbf{N}$, denoting by $x, y, \ldots$ the elements of $\mathbf{V}$, and by $u, v, \ldots$ the elements of $\mathbf{N} \setminus \mathbf{V}$. The faithful π-net which corresponds to the λ-term a is denoted by $, _u(a)$.

Definition 16 *The translation of the lazy λ-calculus into the faithful π-nets is defined inductively by*

$$
\begin{aligned}
&, _u(x) = \leftarrow xu \\
&, _u(\lambda x.a) = u.x[(\leftarrow xs)[\#/s] \otimes u.v[, _v(a)]] \\
&, _u(ab) = (, _v(a) \otimes \\
&\qquad (\leftarrow vx \otimes \\
&\qquad\quad (x.r[\leftarrow au] \otimes \\
&\qquad\qquad a.t[\leftarrow vt \otimes x : w[, _w(b)]] \\
&\qquad\quad)[\#/a] \\
&\qquad)[\#/x] \\
&\quad)[\#/v]
\end{aligned}
$$

The following result shows the correctness of this translation.

Proposition 6 *Let a be a closed λ-term (i.e. without free variables). Then we have one of the following two situations:*

- $a \rightarrow (\lambda y.a')\{b_1/x_1, \ldots, b_n/x_n\}$, *and*
  $, _u(a) \rightarrow (\ldots((, _u(\lambda y.a') \otimes x_1 : w[, _w(b_1)] \otimes \ldots$
  $\ldots \otimes x_n : w[, _w(b_n)])[\#/x_n]) \ldots)[\#/x_1] \not\rightarrow$
- *both* a *and* $, _u(a)$ *diverge (they are divergent).*

Proof: **1** *We use the fact that the translation from faithful π-nets to the π-calculus processes given by* **write** *is sound (proposition 21). The proof is quite similar to the proof given in [Mil92] for the translation of the lazy λ-calculus into π-calculus.*

6 Conclusion

This paper provides a faithful graphical representation of the π-calculus, given by so called faithful π-nets. We show that the faithful π-nets and the π-calculus have the same expressive power, and we define two "fully abstract" translations - from π-calculus to faithful π-nets, and back. These encodings are "fully abstract" when two source calculus terms are equivalent if and only if their translations are equivalent. In order to show how these results are actually related to the operational semantics, a connection between reductions over terms and reductions over their encodings is given by the operational correspondence results for both translations.

We are thinking to extend the study on these faithful π-nets. One direction is to avoid the pseudo-arrows - called connections - by an inheritance mechanism. This step leads to a new calculus for concurrent objects which is somehow close to the ideas reflected by the attempts given by America, Honda, and Vasconcelos on object-oriented concurrent languages [Ame89, HT91, Vas94]. An encoding of a concurrent object-based language into the π-calculus is given in [Wal95].

References

[Ame89] Pierre America. Issues in the design of a parallel object-oriented language, *Formal Aspects of Computing* 1(4), pages 366-411, 1989.

[Abr89] Samson Abramsky. The lazy lambda calculus. In D. Turner(ed.): *Research Topics in Functional Programming*, pages 65-116, Addison-Wesley, 1989.

[BB92] Gérard Berry, Gérard Boudol. The chemical abstract machine, *Theoretical Computer Science* vol.96, pages 217-248, 1992.

[Bou92] Gérard Boudol. Asynchrony and the π-calculus, Technical Report, INRIA Sophia-Antipolis, 1992.

[CR96] Gabriel Ciobanu, Mihai Rotaru. The refinement of the concurrent processes using π-calculus and π-nets, *Proceedings 10th Romanian Symposium on Computer Science*, University Press, pages 241-252, 1996.

[HT91] Kohei Honda, Mario Tokoro. An object calculus for asynchronous communication. In *Proceedings European Conference on Object-Oriented Programming*, LNCS 512, pages 133-147, Springer-Verlag, 1991.

[Mil91] Robin Milner. The polyadic π-calculus: a tutorial, ECS-LFCS Report Series 91-180, Laboratory for Foundations of Computer Science, University of Edinburgh, 1991.

[Mil92] Robin Milner. Functions as processes, *Journal of Mathematical Structures in Computer Science*, vol.2(2), pages 119-141, 1992.

[MPW92] Robin Milner, Joachim Parrow, David Walker. A calculus of mobile processes. *Journal of Information and Computation*, vol.100, pages 1-77, 1992.

[Mil93a] Robin Milner. Action calculi, or syntactic action structures, *Mathematical Foundations of Computer Science*, LNCS, Springer-Verlag, 1993.

[Mil93b] Robin Milner. Action structure for the π-calculus, *Proceedings of the NATO Summer School on Logic and Computation*, Springer-Verlag, 1993.

[Mil94] Robin Milner. π-nets: a graphical form of π-calculus, ESOP'94, LNCS 788, pages 26-42, Springer-Verlag, 1994.

[MP95] Ugo Montanari, Marco Pistore. Concurrent semantics for the π-calculus, *Electronic Notes in Theoretical Computer Science* vol.1, 19 pages, 1995.

[Par93] Joachim Parrow. Interaction diagrams, *A Decade of Concurrency, REX School& Symposium*, LNCS 803, pages 477-508, Springer-Verlag, 1993.

[San92] Davide Sangiorgi. Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms(PhD thesis, ECS-LFCS-93-266, Department of Computer Science, University of Edinburgh, 1993.

[San93] Davide Sangiorgi. From π-calculus to Higher-Order π-calculus – and back. In TAPSOFT'93, LNCS 668, pages 151-166, Springer-Verlag, 1993.

[San94] Davide Sangiorgi. The Lazy Lambda Calculus in a Concurrency Scenario. *Journal of Information and Computation*, vol.111, pages 120-153, 1994.

[Vas94] Vasco Vasconcelos. Typed concurrent objects, In 8th European Conference on Object-Oriented Programming, LNCS 821, pages 100-117, Springer-Verlag, 1994.

[Wal95] David Walker. Objects in the π-calculus, *Journal of Information and Computation*, vol.116, pages 253-271, 1995.

[Yos94] Nobuko Yoshida. Graph notation for concurrent combinators, *Proceedings Theory and Practice of Parallel Programming (TPPP'94)*, LNCS 907, pages 393-412, Springer-Verlag, 1994.

Tree Morphisms and Bisimulations

Rocco De Nicola[1] *Anna Labella*[2]
[1]Dipartimento di Sistemi e Informatica, Università di Firenze
e-mail: denicola@dsi.unifi.it
[2]Dipartimento di Scienze dell'Informazione, Università di Roma "La Sapienza"
e-mail: labella@dsi.uniroma1.it

Abstract

A category of (action labelled) trees is defined that can be used to model unfolding of labelled transition systems and to study behavioural relations over them. In this paper we study five different equivalences based on bisimulation for our model. One, that we called *resource bisimulation*, amounts essentially to three isomorphism. Another, its weak counterpart, permits abstracting from silent actions while preserving the tree structure. The other three are the well known strong, branching and weak bisimulation equivalence. For all bisimulations, but weak, canonical representatives are constructed and it is shown that they can be obtained via enriched functors over our categories of trees, with and without silent actions. Weak equivalence is more problematic; a canonical minimal representative for it cannot be defined by quotienting our trees. The common framework helps in understanding the relationships between the various equivalences and the results provide support to the claim that branching bisimulation is the natural generalization of strong bisimulation to systems with silent moves and that resource and weak resource have an interest of their own.

1 Introduction

Behavioural equivalences play an important rôle in the description of the operational semantics of concurrent systems. These equivalences are used to abstract from the irrelevant details introduced when describing systems as sets of states that evolve by performing actions, i.e. by means of labelled transition systems. There are various opinions about which features of a system are relevant for a given purpose, and hence various notions of equivalence for labelled transition systems have been proposed; [3] and [8, 9] give a comparative accounts.

Many of the equivalences proposed in the literature are based on the notion of *bisimulation* [15] which gives rise to *strong bisimulation equivalence*. Two of the most popular generalizations of this equivalence to systems with silent moves are *branching* [10] and *weak* bisimulation [14]. Both of these equivalences ignore τ (internal or silent) actions, but they deal differently with intermediate states accessed by τ transitions and lead to different identifications, putting a different stress on the branching structure of processes. While strong bisimulation is generally regarded as the equivalence which provides the minimum abstraction from the details of behaviour for $\tau-$free transitions systems, there is little agreement about the comparative merits of the weak and branching generalizations of strong equivalence to systems with τ actions.

In our view, category theory and their abstract constructions can be a useful tool for understanding and assessing the relative merits of different concepts. Here, we consider those categories of trees that have been used to model concurrent systems [12] and nondeterministic regular expressions [6, 2]. Our aim is that of reconducting the different bisimulations to a common framework where it is easier to understand their relationships.

We shall study five different bisimulation-based equivalences for our trees. The more concrete one, that we called *resource bisimulation* [6], corresponds to tree isomorphism; it is finer than strong bisimulation in that it discriminates also according to the number of computations that can be performed to reach specific states. It will be our touchstone and will guide toward defining

and assessing the other equivalences. Indeed, the second one is its weak counterpart that permits abstracting from silent action while preserving the tree structure. The other three equivalences are the well known strong, branching and weak bisimulation equivalence [15, 10, 14]. We shall see that branching bisimulation can be obtained by mirroring the construction for weak resource bisimulation while replacing isomorphism requirements with requirements of strong bisimilarity. For resource, strong, weak resource and branching bisimulation equivalence we shall define standard representatives of their equivalence classes. These constructions are then vindicated by the enriched categorical account that we provide in the final part of the paper. We argue that similar results cannot be obtained for weak bisimulation equivalence.

We start from a basic category of trees labelled over an actions monoid, **Tree**, and construct a category **Der** with the same objects and where the maps are paths to derivatives (so a map $f : t \to t'$ tells us how to find a copy of t' in t). Invisible actions (τs) are introduced by admitting them as labels; this generalizes **Tree** to $\mathbf{Tree}_\tau$ and **Der** to $\mathbf{Der}_\tau$. To relate the two categories a functor $del : \mathbf{Tree}_\tau \to \mathbf{Tree}$ is defined which deletes τ–labelled branches.

We show that resource equivalence does not introduce any quotienting on **Der** and corresponds to the identity functor. After that, we introduce $F_S : \mathbf{Der} \to \mathbf{Der}$ which functorially maps a tree to the canonical representative of its strong bisimulation equivalence class. Finally, we define F_{WR} and $F_B : \mathbf{Der}_\tau \to \mathbf{Der}_\tau$ which functorially maps a tree to the canonical representative of its weak resource and branching bisimulation equivalence class.

In our view, these results strengthen the claim that branching bisimulation is the natural generalization of strong bisimulation to systems with silent moves and that a suitable notion of tree is fundamental in dealing with bisimulations.

2 Trees and Transitions Systems

We begin by introducing the basic concepts of labelled transition systems, their unfoldings, and the five notions of bisimulation we will study. We then present trees as a structure of *runs* with *agreements* and the relationship with unfoldings.

We begin with standard definitions about transition systems. We suppose that a set A of actions is given, together with a distinguished action $\tau \notin A$ representing a silent move. Unless otherwise stated, we confine attention to reachable transition systems with finite unfoldings.

Notation 2.1 We write A_τ for $A \cup \{\tau\}$ and A^* for the monoid of words on A with empty word ϵ. The variables a, b etc. will range over A, and μ, ν etc. over A_τ. Words will be w, v etc.

Definition 2.2 A (rooted) *labelled transition system* or LTS is a quadruple $\mathcal{S} = (S, E, \to, s_0)$ where S is a set of *states*, ranged over by s, u etc.; E is a set of *actions*, $E \subseteq A_\tau$; $\to \subseteq S \times E \times S$ is a relation, the *transition relation*; $s_0 \in S$ is a distinguished *starting state*.

Notation 2.3 We usually write $s \xrightarrow{\mu} s'$ rather than $(s, \mu, s') \in \to$, and if s, u, s' and u' are states in S we write

(i) $\stackrel{\epsilon}{\Longrightarrow}$ for the reflexive and transitive closure of $\xrightarrow{\tau}$;

(ii) $s \stackrel{\mu}{\Longrightarrow} u$ if there exist s', u' such that $s \stackrel{\epsilon}{\Longrightarrow} s' \xrightarrow{\mu} u' \stackrel{\epsilon}{\Longrightarrow} u$;

(iii) $s \stackrel{\mu}{\not\longrightarrow}$ if there exists no $s' \in S$ such that $s \xrightarrow{\mu} s'$;

(iv) $s \not\longrightarrow$ if $s \stackrel{\mu}{\not\longrightarrow}$ for all $\mu \in A_\tau$.

We now introduce our five bisimulations, four of them are relatively well known, the other, weak resource bisimulation, is new and has been defined in collaboration with Flavio Corradini.

Definition 2.4 Let $\mathcal{S} = (S, E, \to, s_0)$ be an LTS. A symmetric relation $R \subseteq S \times S$ is said to be

(i) a *resource bisimulation* if $s\ R\ u$, implies that there exists a bijection f between $\{s'|s \xrightarrow{\mu} s'\}$ and $\{u'|u \xrightarrow{\mu} u'\}$ such that $s'\ R\ f(s')$;

(ii) a *strong bisimulation* if $s\ R\ u$ and $s \xrightarrow{\mu} s'$ implies that $u \xrightarrow{\mu} u'$ and $s'\ R\ u'$;

(iii) a *weak resource bisimulation* if for all $\mu \in A_\tau$, if $s\ R\ u$, then there exists a bijection f between $\{s'|s \stackrel{\mu}{\Longrightarrow} s'\} - \{s'|s' \stackrel{\tau}{\Longrightarrow} s'', s'\ R\ s''\}$ and $\{u'|u \stackrel{\mu}{\Longrightarrow} u'\} - \{u'|u' \stackrel{\tau}{\Longrightarrow} u'', u'\ R\ u''\}$ such that $s'\ R\ f(s')$;

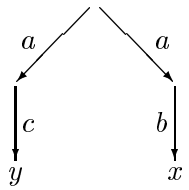
(iv) a *branching bisimulation* if $s\ R\ u$ and $s \xrightarrow{\mu} s'$ implies that either $\mu = \tau$ and $s'\ R\ u$. or $u \stackrel{\epsilon}{\Longrightarrow} u_1 \xrightarrow{\mu} u_2 \stackrel{\epsilon}{\Longrightarrow} u_3$ and $s\ R\ u_1$, $s'\ R\ u_2$, $s'\ R\ u_3$.

(v) a *weak bisimulation* if $s\ R\ u$ and $s \stackrel{a}{\Longrightarrow} s'$ implies that $u \stackrel{a}{\Longrightarrow} u'$ with $s'\ R\ u'$.

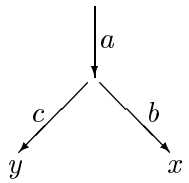
Two states are said to be *resource*, *strong*, *weakly resource*, *branching* or *weak bisimilar* if there exists an eponymous bisimulation relating them. We write $\approx_R$, $\approx_S$, $\approx_{WR}$, $\approx_B$, and $\approx_W$ for *resource*, *strong*, *weak resource*, *branching*, and *weak* bisimulation equivalence, respectively.

It is not difficult to see that, in presence of τ-actions, the last three relations are increasingly coarser. When all actions are visible, we have instead that *resource* and *weak resource* on one hand, and *strong*, *branching* and *weak* bisimulation, on the other, do collapse.

We now introduce a category of labelled trees, **Tree** and some of its properties. A single tree will be modelled by specifying what runs it has, what computations are performed along each run (its *extent*), and to what extent those computations agree (the *agreement*). Thus the tree



will be modelled by runs, x and y, labelled with ab and ac respectively, and by stating that x and y do not agree at all. In contrast, the tree



will be modelled by giving two runs, x and y, again labelled with ab and ac, but with agreement between x and y being the initial a.

Definition 2.5 Let $\mathcal{A} = (A^*, \leq, \wedge, \epsilon)$ be the meet semilattice where $\leq$ is the prefix order of words; $\wedge$ is the largest common prefix operation on words; and ϵ is the minimum.

Definition 2.6 A tree $\mathcal{X} = (X, \varepsilon, \alpha)$ comprises:

- a set X of *runs*;
- a map $\varepsilon : X \to A^*$, the *extent map*, giving the computation $\varepsilon(x)$ along a run x;
- a map $\alpha : X \times X \to A^*$, establishing the *agreement* between pairs of computations.

Additionally, we require that for all $x, y, z \in X$

$$\alpha(x,x) = \varepsilon(x) \qquad \alpha(x,y) \leq \varepsilon(x) \wedge \varepsilon(y) \qquad \alpha(x,y) \wedge \alpha(y,z) \leq \alpha(x,z) \qquad \alpha(x,y) = \alpha(y,x)$$

These amount to requiring that a run agrees with itself along all its length; the agreement between two runs is not bigger than their largest common prefix (runs are forced to agree on a common initial segment and they cannot join up again once split); the common agreement between x, y and z is not bigger than that between x and z; agreement is symmetrical.

We will write $\mathcal{X}$, $\mathcal{Y}$, etc. for typical trees with components $\mathcal{X} = (X, \varepsilon, \alpha)$, $\mathcal{Y} = (Y, \zeta, \beta)$. We shall use $w' - w$ to denote the word obtained from w' by deleting the prefix w from w'.

Example 2.7 The two trees illustrated above are specified by (X, ε, α) where $X = \{x, y\}$, and $\varepsilon(x) = ab$, $\varepsilon(y) = ac$, $\alpha(x,y) = \epsilon$ and (Y, ζ, β) where $Y = \{x, y\}$, and $\zeta(x) = ab$, $\zeta(y) = ac$, $\beta(x,y) = a$, respectively.

We can observe that A-labelled trees are symmetric $\mathcal{A}$–categories, when $\mathcal{A}$ is thought of as a posetal 2-category [16]. Therefore the appropriate notion of comparison for trees is that of $\mathcal{A}$–functor, i.e.:

Definition 2.8 A *tree morphism* $f : \mathcal{X} \to \mathcal{Y}$ is a map $f : X \to Y$ satisfying

(i) f does not change extent: $\zeta(f(x)) = \varepsilon(x)$;

(ii) f increases agreement: $\beta(f(x), f(y)) \geq \alpha(x,y)$.

Tree will be the category of finite A–labelled trees. For A_τ–labelled trees, we use the semi-lattice $\mathcal{A}_\tau = (A_\tau^*, \leq, \wedge, \epsilon)$ exactly as before, and hence obtain the category $\mathbf{Tree}_\tau$ with extent and agreement maps valued in $\mathcal{A}_\tau$.

Definition 2.9 Given two trees, $\mathcal{X}$ and $\mathcal{Y}$, we can form the sequential composition of $\mathcal{X}$ and $\mathcal{Y}$, $\mathcal{X} \otimes \mathcal{Y} = (Z, \eta, \gamma)$ as follows:

(i) $Z = X \times Y$ (a run in $\mathcal{X} \otimes \mathcal{Y}$ is a run in $\mathcal{X}$ followed by a run in $\mathcal{Y}$);

(ii) $\eta(x,y) = \varepsilon(x).\zeta(y)$, where . is concatenation of strings (so the label of a run in $\mathcal{X} \otimes \mathcal{Y}$ is the label in $\mathcal{X}$ followed by the label in $\mathcal{Y}$);

(iii) $\gamma((x,y),(x',y'))$ is $\alpha(x,x')$ if $x \neq x'$ and $\varepsilon(x)\beta(y,y')$ otherwise (so runs that are different in $\mathcal{X}$ have their $\mathcal{X}$ agreement, while runs that differ only in $\mathcal{Y}$ have their $\mathcal{X}$ agreement concatenated with their $\mathcal{Y}$ agreement).

Proposition 2.10 Sequential composition defines the object part of an associative tensor product on **Tree** with unit $\mathbf{1} = (\{\bullet\}, \varepsilon(\bullet) = \epsilon, \alpha(\bullet,\bullet) = \epsilon)$. **Tree** has an initial object given by the empty tree, $\mathbf{0} = (\emptyset, \emptyset, \emptyset)$, and finite coproducts given by joining two trees at the root.

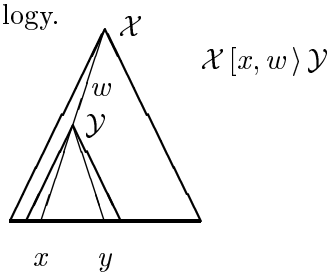
Intuitively, it is clear that the trees introduced in the last sections can be used to represent the unfoldings of finite transition systems. Here, we formally establish the correspondence and use it to motivate our equivalences and lift the bisimulations to trees.

Definition 2.11 Let $\mathcal{X} = (X, \varepsilon, \alpha)$ be a tree. A *prefix of a run* in $\mathcal{X}$ is a pair (x, w) consisting of a run $x \in X$ and a word $w \in A^*$ with $w \leq \varepsilon(x)$.

A *path* in $\mathcal{X}$ is an equivalence class $[x, w]$ of prefixes quotiented by
$(x, w) \equiv (y, v)$ iff $w = v \leq \alpha(x,y)$;

The *derivative* reached along x after w, for a path $[x, w]$ in $\mathcal{X}$, is the tree (Y, ζ, β) where $Y = \{x' \mid x' \in X, \alpha(x,x') \geq w\}$; $\zeta(x') = \varepsilon(x') - w$; $\beta(x', x'') = \alpha(x', x'') - w$.

The figure below illustrates the terminology.



We will write paths($\mathcal{X}$) for the paths of a tree $\mathcal{X}$ and $\mathcal{X}\,[x, w\rangle\,\mathcal{Y}$ if $\mathcal{Y}$ is the derivative reached along x after w in $\mathcal{X}$ We will write $\mathcal{X}\,[x, v\rangle$ for the unique tree $\mathcal{Y}$ such that $\mathcal{X}\,[x, v\rangle\,\mathcal{Y}$.

Notice that in the definition above, we did not mention nodes explicitly: they are in an obvious bijective correspondence with paths. Sometimes, we will refer to paths as nodes.

Given a transition system $\mathcal{S}$ with a finite unfolding, we now construct a tree unf($\mathcal{S}$) representing that unfolding.

Definition 2.12 The *unfolding* $\mathrm{unf}(\mathcal{S}) = (\mathrm{runs}(\mathcal{S}), \varepsilon_{\mathcal{S}}, \alpha_{\mathcal{S}})$ of a transition system $\mathcal{S} = (S, E, \rightarrow, s_0)$ is a tree given by

(i) $\mathrm{runs}(\mathcal{S}) = \{s_0\mu_1 s_1\mu_2 \ldots \mu_n s_n \mid s_0 \xrightarrow{\mu_1} s_1 \xrightarrow{\mu_2} \ldots \xrightarrow{\mu_n} s_n \not\rightarrow\}$;

(ii) $\varepsilon_{\mathcal{S}}(s_0\mu_1 s_1 \ldots \mu_n s_n) = \mu_1 \ldots \mu_n$;

(iii) $\alpha_{\mathcal{S}}(s_0\mu_1 s_1 \ldots \mu_n s_n, s_0\nu_1 u_1 \ldots \nu_m u_m) = \mu_1 \ldots \mu_l$ where for all $k \leq l$, $\nu_k = \mu_k$ and $s_k = u_k$, and $\mu_{l+1} \neq \nu_{l+1}$ or $s_{l+1} \neq u_{l+1}$.

It is not difficult to see that, if we use unfold($\mathcal{S}$) to indicate the standard unfolding of a transition system $\mathcal{S}$ and use tran($\mathcal{X}$) to refer to the transition system associated to a tree $\mathcal{X}$ defined as follows: $\mathrm{tran}(\mathcal{X}) = (\mathrm{paths}(\mathcal{X}), \mathrm{im}(\varepsilon), \rightarrow, [x, \epsilon])$ where $\mathrm{im}(\varepsilon) = \{\mu | \mu \text{appears somewhere in } \varepsilon(x), x \in X\}$ and $[x, w] \xrightarrow{\mu} [y, v]$ iff $(y, v) \in [x, w\mu]$, then we have: $\mathrm{unfold}(\mathcal{S}) \cong \mathrm{tran}(\mathrm{unf}(\mathcal{S}))$.

3 Bisimulation for Trees

We shall work directly on trees and provide a concrete definition of resource and strong equivalence directly over them. We will also consider weak resource equivalence, branching and weak equivalence. We will however prove that our definitions are in full agreement with the corresponding ones introduced in the previous section for labelled transition systems.

Definition 3.1 Two trees, $\mathcal{X}$ and $\mathcal{Y}$, are *resource bisimilar*, written $\mathcal{X} \approx_R \mathcal{Y}$ iff there exists a bijective function $f : X \longrightarrow Y$ such that $\varepsilon(x) = \zeta(f(x))$ and $\forall w \leq \varepsilon(x)$, with $w \neq \epsilon$, $\mathcal{X}\,[x, w\rangle \approx_R \mathcal{Y}\,[f(x), w\rangle$.

Proposition 3.2 Two (finite) trees are resource bisimilar if and only if they are isomorphic.

Proposition 3.3 Two transition systems with finite unfoldings are resource bisimilar, $\mathcal{S} \approx_R \mathcal{S}'$, iff there is a resource bisimulation between their unfoldings as trees, i.e. iff $\mathrm{unf}(\mathcal{S}) \approx_R \mathrm{unf}(\mathcal{S}')$.

Definition 3.4 Two trees, $\mathcal{X}$ and $\mathcal{Y}$, are *strongly bisimilar*, written $\mathcal{X} \approx_S \mathcal{Y}$ iff

i. $\forall x \in X\ \exists y \in Y$: $\varepsilon(x) = \zeta(y)$ and for all $w \leq \varepsilon(x)$, with $w \neq \epsilon$, $\mathcal{X}\,[x, w\rangle \approx_S \mathcal{Y}\,[y, w\rangle$.

ii. $\forall y \in Y\ \exists x \in X$: $\varepsilon(x) = \zeta(y)$ and for all $w \leq \zeta(y)$, with $w \neq \epsilon$, $\mathcal{Y}\,[y, w\rangle \approx_S \mathcal{X}\,[x, w\rangle$.

Proposition 3.5 Two transition systems with finite unfoldings are strongly bisimilar, $\mathcal{S} \approx_S \mathcal{S}'$, iff there is a strong bisimulation between their unfoldings as trees, i.e. iff $\mathrm{unf}(\mathcal{S}) \approx_S \mathrm{unf}(\mathcal{S}')$.

We introduce a function, *del*, which deletes τs, and transforms a tree with τ moves in $\mathbf{Tree}_\tau$ into a tree in **Tree** obtained by ignoring all τ moves. Below, we overload notation, and call *del* the obvious deletion on words, $del(\epsilon) = \epsilon$ and $del(\mu w)$ as $\mu del(w)$ if $\mu \neq \tau$ and $del(w)$ otherwise.

Definition 3.6 Function $del : Tree_\tau \to Tree$ is defined as $del(X, \varepsilon, \alpha) = (Y, \zeta, \beta)$ where $Y = X$; $\zeta(x) = del(\varepsilon(x))$; $\beta(x, y) = del(\alpha(x, y))$.

It can be immediately seen that *del* extends to functor $del : \mathbf{Tree}_\tau \to \mathbf{Tree}$; indeed morphisms from $\mathcal{X}$ to $\mathcal{Y}$ induce morphisms from $del(\mathcal{X})$ to $del(\mathcal{Y})$. Please notice that images of glued runs in $del(\mathcal{X})$ are glued in $del(\mathcal{Y})$.

Once we ignore τs, a derivative is not uniquely determined by its access path any longer. To see this, examine Figure 1: the same run, x, leads to both the derivative $t + \tau t'$ and t' along $del(w)$ or $del(w\tau)$. Thus, in general, $(x, del(w))$ specifies the path to a set of derivatives.

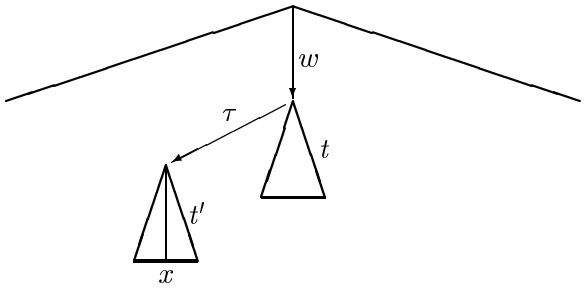


Figure 1: Derivatives accessed along $[x, del(w)]$ in the presence of τs.

It is important to notice that there is always a largest tree ($t + \tau t'$ here) to which $[x, del(w)]$ leads, and any other tree so accessed (like t') is a τ–summand of this one.

Definition 3.7 Given a tree $\mathcal{X} = (X, \varepsilon, \alpha)$ in $\mathbf{Tree}_\tau$, a run $x \in X$, and a prefix $v \leq del(\varepsilon(x))$, let $w \in A_\tau^*$ be the shortest word such that $w \leq \varepsilon(x)$ and $del(w) = v$, $\mathcal{X}\,[x, w\rangle\,\mathcal{Y}$.

1. $\mathcal{R}_\tau(\mathcal{X}, x, v) = \{\mathcal{Z} \mid \mathcal{X}\,[x, w\rangle\,\mathcal{Y}\ [x, \tau^n\rangle\,\mathcal{Z}, n \geq 0\}$,
   the family of derivatives reachable along x by v.
2. $\mathcal{R}(\mathcal{X}, x, v) = \{del(\mathcal{Z}) \mid \mathcal{X}\,[x, w\rangle\,\mathcal{Y}\ [x, \tau^n\rangle\,\mathcal{Z}, n \geq 0\}$,
   the family of derivatives reachable along x by v but pruned using *del*.

Within this setting, we write $\mathcal{Z} \leq_+ \mathcal{Y}$ if $\mathcal{Z}$ is a τ–summand of $\mathcal{Y}$, and note that $\mathcal{R}(\mathcal{X}, x, v)$ is linearly ordered by the relation induced by $\leq_+$, that with abuse of notation we will write $\leq_+$ as well.

Definition 3.8 Two trees $\mathcal{X}$ and $\mathcal{Y}$ are *weak resource bisimilar*, written $\mathcal{X} \approx_{WR} \mathcal{Y}$, iff there exists a bijection $f : X \longrightarrow Y$, such that $del(\varepsilon(x)) = del(\zeta(f(x)))$ and

a) For all $v = del(w)$, $w \leq \varepsilon(x)$ and $w \neq \epsilon$, if $t \in \mathcal{R}_\tau(\mathcal{X}, x, v)$ then there exists $t' \in \mathcal{R}_\tau(\mathcal{Y}, f(x), v)$ and $t \approx_{WR} t'$;

b) for all $v = del(w)$, $w \leq \varepsilon(y)$ and $w \neq \epsilon$, if $t \in \mathcal{R}_\tau(\mathcal{X}, y, v)$ then there exists $t' \in \mathcal{R}_\tau(\mathcal{Y}, f(y), v)$ and $t \approx_{WR} t'$.

Proposition 3.9 Two transition systems are weak resource bisimilar, $\mathcal{S} \approx_{WR} \mathcal{S}'$, iff there is a weak resource bisimulation between their unfoldings as trees, i.e. iff $\mathrm{unf}(\mathcal{S}) \approx_{WR} \mathrm{unf}(\mathcal{S}')$.

Definition 3.10 Two trees $\mathcal{X}$ and $\mathcal{Y}$ are *branching bisimilar*, written $\mathcal{X} \approx_B \mathcal{Y}$, iff

i. $\forall x \in X\ \exists y \in Y$ such that $del(\varepsilon(x)) = del(\zeta(y))$ and

   a) for all $v = del(w)$, with $w \leq \varepsilon(x)$ and $w \neq \epsilon$, if $t \in \mathcal{R}_\tau(\mathcal{X}, x, v)$ then there exists $t' \in \mathcal{R}_\tau(\mathcal{Y}, y, v)$ such that $t \approx_B t'$;

   b) for all $v = del(w)$, with $w \leq \zeta(y)$ and $w \neq \epsilon$, if $t' \in \mathcal{R}_\tau(\mathcal{Y}, y, v)$ then there exists $t \in \mathcal{R}_\tau(\mathcal{X}, x, v)$ such that $t \approx_B t'$.

ii. $\forall y \in Y\ \exists x \in X$ such that $del(\varepsilon(x)) = del(\zeta(y))$ and

   a) for all $v = del(w)$, with $w \leq \zeta(y)$ and $w \neq \epsilon$, if $t' \in \mathcal{R}_\tau(\mathcal{Y}, y, v)$ then there exists $t \in \mathcal{R}_\tau(\mathcal{X}, x, v)$ such that $t \approx_B t'$;

   b) for all $v = del(w)$, with $w \leq \varepsilon(x)$ and $w \neq \epsilon$, if $t \in \mathcal{R}_\tau(\mathcal{X}, x, v)$ then there exists $t' \in \mathcal{R}_\tau(\mathcal{Y}, y, v)$ such that $t \approx_B t'$.

Proposition 3.11 Two transition systems are branching bisimilar, $\mathcal{S} \approx_B \mathcal{S}'$, iff there is a branching bisimulation between their unfoldings as trees, i.e. iff $\text{unf}(\mathcal{S}) \approx_B \text{unf}(\mathcal{S}')$.

Definition 3.12 Two trees $\mathcal{X}$ and $\mathcal{Y}$ are *weakly bisimilar*, written $\mathcal{X} \approx_W \mathcal{Y}$, iff

i. $\forall x \in X\ \exists y \in Y$ such that $del(\varepsilon(x)) = del(\zeta(y))$ and

   a) for all $v = del(w)$, with $w \leq \varepsilon(x)$ and $w \neq \epsilon$, if $t \in \mathcal{R}_\tau(\mathcal{X}, x, v)$ then there exists $t' \in \mathcal{R}_\tau(\mathcal{Y}, y, v)$ such that $t \approx_W t'$.

ii. $\forall y \in Y\ \exists x \in X$ such that $del(\varepsilon(x)) = del(\zeta(y))$ and

   a) for all $v = del(w)$, with $w \leq \zeta(y)$ and $w \neq \epsilon$, if $t' \in \mathcal{R}_\tau(\mathcal{Y}, y, v)$ then there exists $t \in \mathcal{R}_\tau(\mathcal{X}, x, v)$ such that $t \approx_W t'$.

Proposition 3.13 Two transition systems are weakly bisimilar, $\mathcal{S} \approx_W \mathcal{S}'$, iff there is a weak bisimulation between their unfoldings as trees, i.e. iff $\text{unf}(\mathcal{S}) \approx_W \text{unf}(\mathcal{S}')$.

It is interesting to note the essential difference between definitions 3.10 and 3.12 is one of symmetry: definition 3.12 is missing cases i.b and ii.b of definition 3.10. This will turn out to have important consequences; the symmetrical form of definition 3.10 means that we can define a branching bisimulation as an equivalence relation between runs (in the style of the 'back–and–forth' approach [7]), whereas no such definition will be possible for weak bisimulation.

4 Canonical Representatives for Bisimulation

In the previous section, we rephrased the definition of weak and branching bisimulation by only relying on τ–less structures. The information about silent step transitions is collected in what we called the family of derivatives reached along run x via label w, $\mathcal{R}(\mathcal{X}, x, w)$. In this section we will exploit this intuition, and the construction of standard representatives for strong equivalence classes of trees, to build standard representatives for branching equivalence classes of trees. We will show that, given a rigid (τ–less) tree whose nodes are labelled by $\mathcal{R}$–sets, it is possible to obtain a (non–rigid) "minimal" tree that is branching equivalent to the original one. The same procedure will be used for weak resource equivalence, taking into account that, in that case, the corresponding "rigid" equivalence is isomorphism. We remind the reader that we use $\cong$ to denote tree isomorphism.

We begin by characterizing strong bisimulation via a canonical representative. The canonical representative for the $\approx_S$–equivalence class will be obtained by merging those runs that have the same extent and equivalent relationships with other runs in the same tree.

Definition 4.1 Let $\equiv$ be the equivalence relation on runs defined by $x \equiv x'$ iff $\varepsilon(x) = \varepsilon(x')$ and for every $v \leq \varepsilon(x)$, $\mathcal{X}[x, v\rangle \approx_S \mathcal{X}[x', v\rangle$, and let $|x|$ denote the $\equiv$–equivalence class of x. The *canonical* $\mathbf{S}$*–reduction* of a tree $\mathcal{X} = (X, \varepsilon, \alpha)$, $\mathbf{S}\mathcal{X}$, is the tree (Y, ζ, β) where

(i) $Y = \{|x| \mid x \in X\}$;

(ii) $\zeta(|x|) = \varepsilon(x)$;

(iii) $\beta(|x|, |y|) = \max\{\alpha(x', y') \mid x' \in |x|, y' \in |y|\}$.

It is worth noticing that the maximum above exists. Indeed, $x \equiv x'$ implies $\varepsilon(x) = \varepsilon(x')$; hence $\alpha(x', y)$ for $x' \in |x|$ is always a prefix of $\varepsilon(x)$. Thus $\alpha(x', y)$, as x' varies, is linearly ordered.

We approach the proof that $\mathbf{S}\mathcal{X}$ is the canonical representative of the $\approx_S$–equivalence class of $\mathcal{X}$ (and hence that strong equivalence coincides with $\approx_S$) by showing that $\mathcal{X}$ and $\mathbf{S}\mathcal{X}$ have the same transitions:

Lemma 4.2 *$\mathcal{X}$ and $\mathbf{S}\mathcal{X}$ can perform the same labelled transitions:*

(i) if $\mathcal{X}[x, w\rangle \mathcal{Y}$ then $(\mathbf{S}\mathcal{X})[|x|, w\rangle (\mathbf{S}\mathcal{Y})$;

(ii) if $(\mathbf{S}\mathcal{X})[|x|, w\rangle \mathcal{Z}$ and $\mathcal{X}[x, w\rangle \mathcal{Y}$, then $\mathbf{S}\mathcal{Y} \cong \mathcal{Z}$.

Theorem 4.3 Given trees, $\mathcal{X}$ and $\mathcal{Y}$, we have $\mathbf{S}\mathcal{X} \cong \mathbf{S}\mathcal{Y}$ if and only if $\mathcal{X} \approx_S \mathcal{Y}$.

In order to define a canonical representative for equivalences involving silent moves, we need a procedure of reconstruction of a non rigid tree starting from data given in terms of rigid trees. In the appendix we report an example reconstruction, here we provide a general procedure that given a collection of (x, v)–indexed families of trees, under some conditions on the collection, yields a reconstructed tree.

Definition 4.4 Fix a tree $\mathcal{X}$ in **Tree** and suppose that a finite collection $\mathcal{R}(x, v)$ of sets of trees is given, one set for each pair (x, v), $x \in X$, $v \leq \varepsilon(x)$ We will call this collection $\mathcal{X}$*–reconstructible* if it satisfies the following conditions:

(i) $(\mathcal{R}(x, v), \leq_+)$ is a finite chain, $R(x, v)_i \leq_+ R(x, v)_{i-1} \leq_+ \ldots \leq_+ R(x, v)_1$, in **Tree**;

(ii) There exists a surjective morphism (epimorphism) $f_{x,v}$ from $\mathcal{X}[x, v\rangle$ to the maximal element in the chain $R(x, v)_1$ and for every i, $f_{x,v}(x) \in R(x, v)_i$;

(iii) $f_{x,v}(y) \in R(x, v)_i$ then $\mathcal{R}(x, s) = \mathcal{R}(y, s)$ for all $s < v$, and, if $s = v$, for all $j \leq i$ $R(x, s)_j = R(y, s)_j$, furthermore, for all $s \leq v$, $f_{x,s}$ and $f_{y,s}$ coincide on common domains.

Condition (i) means that the derivative after v along x is considered as standing for i different states that are bigger and bigger, but (ii) guarantees that the biggest of such states is covered by the original one; (iii) deals with coherence of the derivatives associated with each run.

The properties of reconstructible families are sufficient to ensure a well–behaved reconstruction.

Definition 4.5 Given a tree $\mathcal{X}$ in **Tree**, consider a $\mathcal{X}$–reconstructible family $\mathcal{R}(x, v)$, the *reconstruction* $\int \mathcal{X}, \mathcal{R}(x, v) = (\mathcal{X}', \varepsilon', \alpha')$ of $\mathcal{X}$ is given by:

(i) $\mathcal{X}' = \mathcal{X}$;

(ii) $\varepsilon'(x) = \tau^{i_1} a_1 \tau^{i_2} a_2 \ldots \tau^{i_n} a_n \tau^{i_{n+1}}$, given $\varepsilon(x) = a_1 a_2 \ldots a_n$ and $i_k = |\mathcal{R}(x, a_1 \ldots a_{k-1})| - 1$ for $1 \leq k \leq n + 1$;

(iii) $\alpha'(x, y) = \tau^{i_1} a_1 \tau^{i_2} a_2 \ldots \tau^{i_m} a_m \tau^{i_{m+1}}$ where $\alpha(x, y) = a_1 a_2 \ldots a_m$, for $1 \leq k \leq m$, $i_k = |\mathcal{R}(x, a_1 \ldots a_{k-1})| - 1$ and $i_{m+1} = |\mathcal{R}(\mathcal{X}, x, a_1 a_2 \ldots a_m) \cap \mathcal{R}(y, a_1 a_2 \ldots a_m)| - 1$.

Proposition 4.6 Let $\int\ \mathcal{X}, \mathcal{R}(x,v)$ be as in definition 4.5, then it is a tree in $\mathbf{Tree}_\tau$.

Lemma 4.7 Given a tree $\mathcal{X}$ in **Tree**, let w be a word different from τ. There is an epimorphism between $del(\int\ \mathcal{X}, \mathcal{R}(x,v)\left[x, w\tau^h\right\rangle)$ and $R(x, del(w))_{h+1}$.

Proposition 4.8

(i) There is a bijection between $\mathcal{R}_\tau(\int\ \mathcal{X}, \mathcal{R}(x,v))$ and $\mathcal{R}(x,v)$.

(ii) $\int(\mathcal{X}\,[x', w\rangle), \mathcal{R}(x,v)) = (\int\ \mathcal{X}, \mathcal{R}(x,v))\,[x', w\rangle$.

Let us now consider reconstruction in the cases of interest for us. We will start with a tree in $\mathbf{Tree}_\tau$, we apply deletion on it and reconstruct it with families obtained from the original tree. Two kinds of families will be considered, to obtain trees that are weakly resource and branching bisimilar to the original tree. As in the general case (Definition 4.5), reconstruction will be carried run by run and two trees will be identified only if they are isomorphic and reachable along the same run.

Theorem 4.9 Given a tree $\mathcal{X}$ in $\mathbf{Tree}_\tau$, let us consider two $del(\mathcal{X})$–reconstructible families:

(i) $\mathcal{R}_1(x,v) = \mathcal{R}(\mathcal{X}, x, v)$, epimorphisms are given by identity;

(ii) $\mathcal{R}_2(x,v) = \mathbf{S}\ (\mathcal{R}(\mathcal{X}, x, v))$, epimorphisms are functions induced by $\mathbf{S}$.

Then

(i) $\mathcal{X} \approx_{WR} \int\ \mathcal{X}, \mathcal{R}_1(x,v)$ and

(ii) $\mathcal{X} \approx_B \int\ \mathcal{X}, \mathcal{R}_2(x,v) \approx_B \mathbf{S}\int\ \mathcal{X}, \mathcal{R}_2(x,v)$.

Definition 4.10

(i) The *canonical* $\mathbf{WR}$*–reduction* of a tree $\mathcal{X} = (X, \varepsilon, \alpha)$ in $\mathbf{Tree}_\tau$, written $\mathbf{WR}\mathcal{X}$, is $\int \mathcal{X}, \mathcal{R}_1(x,v)$.

(ii) The *canonical* $\mathbf{B}$*–reduction* of a tree $\mathcal{X} = (X, \varepsilon, \alpha)$ in $\mathbf{Tree}_\tau$, written $\mathbf{B}\mathcal{X}$, is $\mathbf{S}\int\ \mathcal{X}, \mathcal{R}_2(x,v)$.

Lemma 4.11 Given a tree $\mathcal{X}$ in $\mathbf{Tree}_\tau$, two derivatives $t, t' \in \mathcal{R}_\tau(\mathcal{X}, x, v)$ are

(i) weak resource bisimilar iff $del(t) = del(t')$;

(ii) branching bisimilar iff $\mathbf{S}\,del(t) \cong \mathbf{S}\,del(t')$.

Theorem 4.12 If $\mathcal{X}$ and $\mathcal{Y}$ are two trees, then we have

(i) $\mathcal{X} \approx_{WR} \mathcal{Y}$ if and only if $\mathbf{WR}\mathcal{X} \cong \mathbf{WR}\mathcal{Y}$.

(ii) $\mathcal{X} \approx_B \mathcal{Y}$ if and only if $\mathbf{B}\mathcal{X} \cong \mathbf{B}\mathcal{Y}$.

Due to theorem 4.12 $\mathbf{WR}\mathcal{X}$ can be thought of as the minimal weak resource representative for $\mathcal{X}$, while $\mathbf{B}\mathcal{X}$ as its minimal branching representative.

We have not been able to provide a standard minimal representation for weak bisimulation by following the pattern of the construction for the other two weak equivalences. The reason for this idiosyncrasy is the impossibility of building the standard representation via quotienting: weak bisimulation does not enforce a direct correspondence between the *runs* of equivalent trees, and hence we cannot build a canonical representative as a quotient over the set of runs of a tree. To see this, consider the two weakly equivalent trees corresponding to the two terms

$$a(\tau b + c) + ab + a(\tau b + d) \qquad \textit{and} \qquad a(\tau b + c) + a(\tau b + d).$$

Now the tree corresponding to the second term is a good candidate for a minimal standard representative. However, the composition of the equivalence class of runs is unclear: the run ab of the first tree can either be absorbed by $a\tau b$ in the first or the third summands, and there is clearly no reason to prefer one choice over the other. Moreover, we cannot put it in both equivalence classes, for that would leave us with $a(\tau b + c + d)$ as the representative, and this is not bisimilar to the original.

5 An Enriched–Categorical Account

In this section we rephrase our account using more explicitly categorical machinery. We will show that the construction of minimal representative is "functorial" w.r.t. tree structure in all the cases. Furthermore a characterization of the resulting functors is given, that emphasizes the fact that the only difference between the resource-weak resource cases and strong-branching cases is the difference between the existence of a bijective function and the existence of relation epimorphic on both sides.

The notion that $\mathcal{Y}$ is a derivative of $\mathcal{X}$ accessed by a word w along a run x, $\mathcal{X}\,[x, w\rangle\,\mathcal{Y}$, naturally leads to a notion of map between trees different from our morphism. Clearly we could define the set of maps between $\mathcal{X}$ and $\mathcal{Y}$, as

$$\{[x, w] \mid \mathcal{X}\,[x, w\rangle\,\mathcal{Y}\}$$

and this would lead to a category of trees where a map from $\mathcal{X}$ to $\mathcal{Y}$ is a way of finding the derivative $\mathcal{Y}$ in $\mathcal{X}$. However, a moment's reflection shows that these arrows from $\mathcal{X}$ to $\mathcal{Y}$ are not just a set: they naturally bear a tree structure. Indeed, there are not just two paths from $\mathcal{X} \otimes \mathcal{Y}$ to $\mathcal{Y}$ of example $\otimes$, there is a tree, consisting of two runs; see Figure 2.

Definition 5.1 The category **Der**, has trees as objects, arrows $f : \mathcal{X} \to \mathcal{Y}$ in $\mathbf{Der}[\mathcal{X}, \mathcal{Y}]$ are paths $[x, w]$ such that $\mathcal{X}\,[x, w\rangle\,\mathcal{Y}'$ where $\mathcal{Y}'$ is an isomorphic copy of $\mathcal{Y}$. Given the tree of paths $\mathbf{Der}[\mathcal{X}, \mathcal{Y}]$ and $\mathbf{Der}[\mathcal{Y}, \mathcal{Z}]$, their composition in **Der** is given by concatenation in **Tree**.

Der is properly seen not just as a category, but as a category enriched over **Tree** equipped with the monoidal structure $\otimes$ [13].

Similarly, $\mathbf{Der}_\tau$ is the category of paths to derivatives with τs defined over $\mathbf{Tree}_\tau$ in the same way as **Der** is defined over **Tree**.

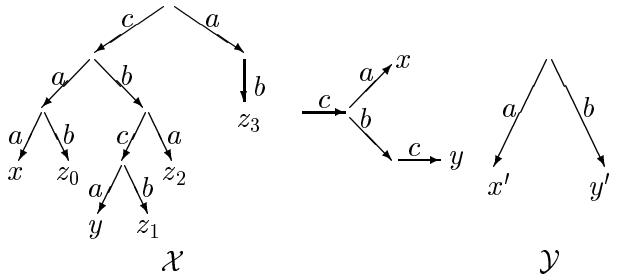


Figure 2: Maps in **Der**: $\mathcal{X}\,[x, ca\rangle\,\mathcal{Y}$, $\mathcal{X}\,[y, cbc\rangle\,\mathcal{Y}$.

We have also that $\mathbf{Der}_\tau$ is, as well, a **Tree**–category due to the effect of the functor $del : \mathbf{Tree}_\tau \to \mathbf{Tree}$ obtained by applying del to homs. In the sequel, $\mathbf{Der}_\tau$ will always denote this **Tree**–category.

Of course the identity functor on **Tree** induces the identity functor on **Der**, but we will show now that the reduction maps **S**, **B** and **WR**, though not being endofunctors, do induce **Tree**–functors from **Der** ($\mathbf{Der}_\tau$) to itself.

Lemma 5.2 Given any two trees, $\mathcal{Z}$ and $\mathcal{Z}'$, there is a **Tree**–map between the trees:

(i) $\mathbf{Der}[\mathcal{Z}, \mathcal{Z}']$ and $\mathbf{Der}[\mathbf{S}\mathcal{Z}, \mathbf{S}\mathcal{Z}']$.

(ii) $del(\mathbf{Der}_\tau[\mathcal{Z}, \mathcal{Z}'])$ and $del(\mathbf{Der}_\tau[\mathbf{WR}\mathcal{Z}, \mathbf{WR}\mathcal{Z}'])$.

(iii) $del(\mathbf{Der}_\tau[\mathcal{Z}, \mathcal{Z}'])$ and $del(\mathbf{Der}_\tau[\mathbf{B}\mathcal{Z}, \mathbf{B}\mathcal{Z}'])$.

Theorem 5.3 The endomap **S** on **Tree** induces a **Tree**–functor, F_S, on **Der**. The endomaps **WR** and **B** on $\mathbf{Tree}_\tau$ induce **Tree**–functors, F_{WR}, and F_B on $\mathbf{Der}_\tau$.

We now go on to examine the nice property that allows us to characterize the **Tree**–functors above.

To begin with, consider the notion of a $\mathcal{V}$–functor $F : \mathcal{C} \to \mathcal{C}$ over some $\mathcal{V}$–category $\mathcal{C}$ being *full* [13]. For this to be the case, we require that for each pair of objects, A, B, the induced function $F_{A,B}$ from $\mathcal{C}[A, B]$ to $\mathcal{C}[FA, FB]$ is an epimorphism. Fullness condition in our case would amount to asking that all paths from FA to FB arise via paths from A to B. This is both too naïve and too demanding.

We want to require that all paths from Ft to any u are obtained via some u_i such that $Fu_i \cong u'$; this is the notion that will allow us to capture functors like ours. intent.

Definition 5.4 A $\mathcal{V}$–functor $F : \mathcal{C} \to \mathcal{C}$ is said to be *hereditarily full* if and only if for any objects A, B of $\mathcal{C}$, there exists a family $\{B_i\}$ such that $FB_i \cong B$ and $\{F_{A,B_i}\}$ covers $\mathcal{C}[FA, B]$.

Proposition 5.5

(i) $F_S : \mathbf{Der} \to \mathbf{Der}$ is an hereditarily full **Tree**–functor.

(ii) $F_{WR} : \mathbf{Der}_\tau \to \mathbf{Der}_\tau$ is an hereditarily full **Tree**–functor preserving $\mathbf{1}$ and sums.

(iii) $F_B : \mathbf{Der}_\tau \to \mathbf{Der}_\tau$ is an hereditarily full **Tree**–functor.

The only trees in **Tree** that have none but trivial derivatives are finite, nonempty, sums of $\mathbf{1}$s. Let us call them *quasiterminals*. If $F(t)$ is a quasi terminal, so is t. If F is hereditarily full, one has the viceversa, i.e. quasi terminals are preserved as a class.

Naturally, identity (the **Tree**–functor $\mathbf{Der} \to \mathbf{Der}$ induced by resource bisimulation) is hereditarily full. It is the only one, up to isomorphism, enjoying this property and preserving $\mathbf{1}$ and sums, hence preserving quasi terminals as individuals. This fact can be easily proved by induction on the depth of the tree. Next theorem will show that the other **Tree**–functors considered in this paper enjoy a similar feature, because they are in some sense universal with respect to the class of **Tree**–functors with the same properties. The statement corresponds to the minimality of the canonical representative.

Theorem 5.6 (i) For all hereditarily full **Tree**–functors $F : \mathbf{Der} \to \mathbf{Der}$, $F_S F \cong F_S$.

(ii) For all hereditarily full, preserving $\mathbf{1}$ and sums, **Tree**–functors $F : \mathbf{Der}_\tau \to \mathbf{Der}_\tau$, $F_{WR} F \cong F_{WR}$.

(iii) For all hereditarily full **Tree**–functors $F : \mathbf{Der}_\tau \to \mathbf{Der}_\tau$, $F_B F \cong F_B$.

A direct consequence of this theorem is that all hereditarily full **Tree**–functors preserving $\mathbf{1}$ and sums, preserve weak resource bisimulation equivalence, while all hereditarily full **Tree**– functors preserve branching bisimulation equivalence.

6 Conclusions

We have studied labelled trees as unfoldings of transitions systems and characterized different bisimulations as special functors between categories of trees, enjoying universal properties. We have thus devised criteria for comparing and assessing different equivalences: branching bisimulation appears as the natural generalization of strong bisimulation just like weak resource bisimulation is the natural generalization of isomorphism of trees.

The definition of the functors has required, as an intermediate step, the construction of a canonical representative of the considered equivalence classes. The construction of canonical representatives for weak bisimulation equivalence turned out to be problematic; we could not define a quotient that preserved the structure of the runs.

Our approach to bisimulations characterizations is related to that introduced in [11] and used in [1], only they start from a different view of the same "topological" structure. Our trees have originally been defined as categories enriched over a locally–posetal 2–category $\mathcal{A}$ namely that associated with the free monoid A^* [12]. Similarly, morphisms between trees are $\mathcal{A}$–functors. It is well known that our trees, as categories enriched on a posetal 2-category, can be thought as presentations of sheaves on the topology where elements of A^* constitute a base. To obtain the corresponding sheaves we would roughly need to complete runs with all their prefixes. To recover the approach followed in [11], elements of A^* could be considered as a subcategory $\mathbf{P}$ of paths in **Tree** and we could characterize strong and branching bisimulation as in [11] via spans of $\mathbf{P}$–open maps. The **Tree**–functoriality corresponds to preservation of "path logic", but our construction, provides also minimal representatives that cannot be obtained via spans. As in our case, characterization of weak equivalence in [11] is problematic, see [1]; it requires introducing an "ad hoc" selection of morphisms or a weakening of the logic to be preserved. This weaker characterization is not reproducible in our context that is more demanding on structural properties.

The two new equivalences that we have considered, and that are not considered in the above mentioned papers, have proved very useful. *Resource bisimulation* has been used to obtain a complete axiomatization of a tree-based interpretation of regular expressions [6] and to provide alternative operational semantics of process algebras [2]. *Weak resource bisimulation* can be fully axiomatized by simply adding to the axioms for resource bisimulation the following law:

$$\alpha;\tau;X \;=\; \alpha;X$$

that essentially says that all and only the "irrelevant" τs are ignored.

Besides this line of investigation, let us mention two promising topics for further work. In this paper we have only considered action–labelled finite trees. There are two obvious generalizations.

Firstly, like it has been done for the open-maps approach, we could export our characterizations to different semantics, those that admit an enriched categorical presentation, i.e. we could consider richer labels that would enable us to rely on the same bisimulations also for non-interleaving models of concurrency [17] and capture, e.g., causal dependence, maximal concurrency, locality–based properties in the same vein of [4, 5]. Secondly, we could consider finite state transition systems with cycles (and hence infinite unfoldings).

However, while the generalization to richer labels is direct, the adjustments needed for dealing with infinity are not minor. Indeed, a key point of our approach is that unfoldings of systems are described as sets of runs from an initial to a final state. Now, while in the case of finite LTSs we immediately have final states, in the cyclic case, we would need to single out specific states as final and ensure that all of them are equivalent. One possibility is to 'massage' systems to include sink states in correspondence with each final state, for instance via the (standard automata–theoretic) construction of introducing epsilon moves. The set of runs of an LTS would then be the set of all finite runs with the obvious labeling; the agreement of any two runs would be the string associated with their initial common run. A run x is considered an approximation of a run y whenever $\alpha(x,y) = \varepsilon(x) < \varepsilon(y)$.

But this will be the subject of future research.

References

[1] A. Cheng, M. Nielsen. Open Maps (at) Work. *BRICS Report Series*,n. 23, 1995.

[2] Corradini,F De Nicola,R. and Labella,A. Fully Abstract Models for Nondeterministic Regular Expressions. In Proc. *Concur '95*. LNCS **962**, Springer Verlag 1995, pp. 130-144.

[3] R. De Nicola. Extensional equivalences for transition systems. *Acta Informatica*, **24**, pp. 211–237, 1987.

[4] P. Degano, R. De Nicola and U. Montanari. Universal Axioms for Bisimulation, *Theoretical Computer Science* volume **14**, North Holland, pp. 63–91, 1993.

[5] P. Degano, R. De Nicola and U. Montanari. Observation Trees, In *Proc. North American Conference on Process Algebras*, (Purushothaman,S. and Zwarico, A. eds.) Springer-Verlag, Workshops in Computing series, pp. 103-118, 1993.

[6] De Nicola,R. and Labella,A. A Completeness theorem for Nondeterministic Kleene Algebras. in Proc. *MFCS '94*. LNCS **841**, Springer Verlag 1994, pp. 536-45.

[7] R. De Nicola, U. Montanari, and F. Vaandrager. Back and forth bisimulations. In *CONCUR'90* (J. Baeten and J. Klop eds.) volume **458**, Springer-Verlag LNCS, 1990.

[8] R. van Glabbeek. The linear time - branching time spectrum. In *CONCUR'90* (J. Baeten and J. Klop, eds.), volume **458**, Springer-Verlag LNCS, pp. 278–297, 1990.

[9] R. van Glabbeek. The linear time - branching time spectrum II (The Semantics of sequential systems with silent moves). In *CONCUR'93* (E. Best, ed.), volume **715**, Springer-Verlag LNCS, pp. 66–81, 1993.

[10] R. van Glabbeek and W. Weijland. Branching Time and Abstraction in Bisimulation Semantics *Journal of the ACM*, **43**(3), pp. 555–600, 1996.

[11] A. Joyal, M. Nielsen and G. Winskel. Bisimulations and Open Maps. In *Logic in Computer Science*, pp. 418–427, 1993.

[12] S. Kasangian, A. Labella, and A. Pettorossi. Observers, Experiments, and Agents: A comprehensive approach to parallelism. In *Semantics of Concurrency*, (Guessarian,I. ed.) volume **469**. Springer-Verlag LNCS, pp. 375-406, 1990.

[13] G. Kelly. *Basic Concepts of Enriched Category Theory.* Cambridge University Press, 1982.

[14] R. Milner. *Communication and concurrency.* International series on computer science, Prentice Hall International, 1989.

[15] D. Park. Concurrency and automata on infinite sequences. In *Proceedings of Theoretical Computer Science 1981*, volume 104. Springer-Verlag LNCS, pp. 167–183, 1981.

[16] R. Walters. Sheaves and Cauchy-complete categories. *Cahiers de Topologie et Geometrie Diff.*, **22**, pp. 283–286, 1981.

[17] G. Winskel and M. Nielsen. Categories of models of concurrency. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*. Vol. **4**, Oxford Science Publications, pp. 2–148, 1995.

7 APPENDIX: Rebuilding Trees

In this appendix we provide an example of the reconstruction procedure formally defined in Section 4.

First of all, we show how to obtain decorated rigid trees from those with silent actions.

In Figure 3 we have represented the tree $\mathcal{X}$ and the tree del($\mathcal{X}$) obtained by deleting all silent moves from $\mathcal{X}$. For the sake of readability, we name y_i the runs of $del(\mathcal{X})$ corresponding to the x_i of $\mathcal{X}$.

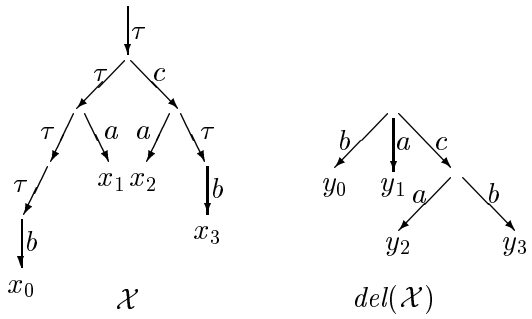


Figure 3: A tree $\mathcal{X}$ in $\mathbf{Tree}_\tau$ and its deletion.

The first step of our reconstruction consists of decorating each node $[y_i, u]$ in $del(\mathcal{X})$ with the derivatives in $\mathcal{R}(\mathcal{X}, x_i, v)$. For instance, we decorate the nodes of $del(\mathcal{X})$ in this way:

(i) the root is decorated with four sets of derivatives, one for each y_i,

$$\begin{aligned}
\mathcal{R}(\mathcal{X}, x_0, \epsilon) &= \{b + a + c(a + b),\ b + a,\ b\} \\
\mathcal{R}(\mathcal{X}, x_1, \epsilon) &= \{b + a + c(a + b),\ b + a\} \\
\mathcal{R}(\mathcal{X}, x_2, \epsilon) &= \{b + a + c(a + b)\} \\
\mathcal{R}(\mathcal{X}, x_3, \epsilon) &= \{b + a + c(a + b)\}
\end{aligned}$$

(ii) the leaf node of the y_0 branch, $[y_0, b]$, is decorated with:

$$\mathcal{R}(\mathcal{X}, x_0, b) = \{\mathbf{1}\}$$

(iii) the leaf node of the y_1 branch is decorated with:

$$\mathcal{R}(\mathcal{X}, x_1, a) = \{\mathbf{1}\}$$

(iv) the node $[y_2, c] = [y_3, c]$ is decorated with:

$$\begin{aligned}
\mathcal{R}(\mathcal{X}, x_2, c) &= \{a + b\} \\
\mathcal{R}(\mathcal{X}, x_3, c) &= \{a + b,\ b\}.
\end{aligned}$$

(v) the leaf node of the y_2 branch is decorated with:

$$\mathcal{R}(\mathcal{X}, x_2, ca) = \{\mathbf{1}\}$$

(vi) the leaf node of the y_3 branch is decorated with:

$$\mathcal{R}(\mathcal{X}, x_3, cb) = \{\mathbf{1}\}$$

The reconstruction of a tree in $\mathbf{Tree}_\tau$ from the decorated version of $del(\mathcal{X})$ proceeds along the following lines.

Given the set of runs x_0, x_1, x_2, x_3, first, their new extent is defined. To do this we rely on the fact that each tree in $\mathcal{R}(\mathcal{X}, x, wa_i)$ represents a derivative accessible by a $\stackrel{a_i}{\Longrightarrow}$–step from $[x, w]$, and reconstruct the extent by introducing after each a_i a number of τs equal to $|\mathcal{R}(\mathcal{X}, x, wa_i)| - 1$ in order to guarantee the necessary branching points. To see this, consider Figure 4, where it is assumed that:

$$\mathcal{R}(\mathcal{X}, z_1, del(wa)) \;=\; \{t4 + t3 + t2 + t1, t3 + t2 + t1, t2 + t1, t1\}$$

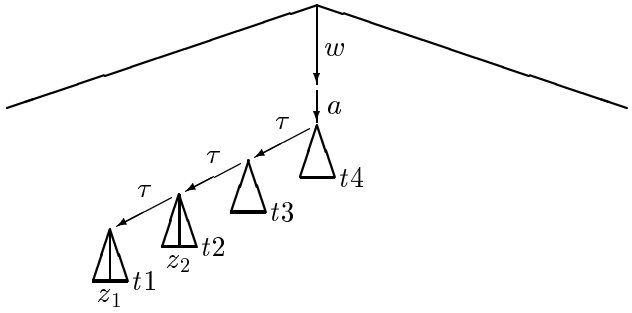


Figure 4: A step in the reconstruction.

In our specific example the suggested construction amounts to defining:

$$\varepsilon(x_0) = \tau^2 b, \qquad \varepsilon(x_1) = \tau a, \qquad \varepsilon(x_2) = ca, \qquad \varepsilon(x_3) = c\tau b$$

The agreement between two given runs is then obtained again by adding after each a_i a number of τs equal to

$$|\mathcal{R}(\mathcal{X}, x_i, wa_i) \cap \mathcal{R}(\mathcal{X}, x_j, wa_i)| - 1$$

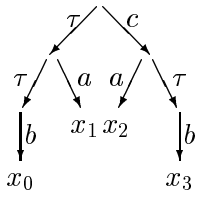


Figure 5: The reconstructed tree: $\int \mathcal{R}(\mathcal{X}, x, v)$.

Thus, the complete reconstruction of the tree of Figure 3, which will be written $\int\ \mathcal{R}(\mathcal{X}, x, v)$, is shown in Figure 5. The reader may like to check that the reconstruction is weak resource bisimilar to the original tree.

Branching Processes of general S/T-Systems

Stefan Haar
Dept. of Computer Science
Humboldt - University of Berlin, Germany
e-mail haar@informatik.hu-berlin.de

June 30, 1998

We introduce an alternative occurrence net semantics for S/T-systems allowing arbitrary markings and arc weights. For Petri net systems, branching process semantics have been introduced by Nielsen, Plotkin, and Winskel [7] and Engelfriet [2]. Branching Processes (BPs for short) give a partial order representation of system behavior by means of an unfolding into an occurrence net. McMillan [6], Esparza [3] and Esparza/Römer/Vogler [5] used BPs to decide system properties in the 1-safe case whilst avoiding state explosion.

Our definition will work for general systems, including those with arc weights greater than one. The restriction to the 1-safe case can therefore be lifted. Also, the principles in constructing the unfolding are different, making the definition more general and, presumably, flexible to include future extensions to different net classes.

We begin by stating the definitions and terminology.

Definition 0.1 *A* **net** *is a quadruple* $\mathcal{N} = (S, T, F, W)$ *such that*

1. S *is a set of* **places**
2. T *a set of* **transitions** *such that* $S \cap T = \emptyset$,
3. $F \subset [(S \times T) \cup (T \times S)]$ *a set of* **arcs**, *and*
4. $W : [(S \times T) \cup (T \times S)] \to \mathbb{N}_0$ *an* **arc weight function** *such that* $W(x, y) = 0$ *iff* $(x, y) \notin F$.

$\mathcal{N}$ *is* **ordinary** *iff* $W(x, y) = 1$ *for all* $(x, y) \in F$.

Adding *markings* and their dynamics, on obtains *systems*:

Definition 0.2 *For any net $\mathcal{N} = (S, T, F, W)$, any mapping $M : S \to \mathbb{N}_0$ is called a* **marking**. *A* **(net) system** *is a pair $\Sigma = (\mathcal{N}, M_0)$ where $\mathcal{N} = (S, T, F, W)$ is a net and $M_0 : S \to \mathbb{N}_0$ a marking, called the* **initial marking** *of Σ. $t \in T$ is* **enabled** *in a marking M iff, for all $s \in S$, $M(s) \geq W(s,t)$. If t is enabled in M, the* **firing** *of t leads to a new marking M', in short: $M[t\rangle M'$, iff, for all $s \in S$, M' satisfies*

$$M'(s) \quad = \quad M(s) - W(s,t) + W(t,s).$$

We denote the set of transitions enabled in M by $\mathcal{ENAB}(M)$, and define the **reachability set** *of M as*

$$\begin{aligned}\mathcal{REACH}(M) \quad := \quad \{\bar{M} : S \to \mathbb{N}_0 : \quad & \exists\, t_1, ..., t_n \in T,\ M_1, ..., M_n : S \to \mathbb{N}_0 : \\ & M[t_1\rangle M_1[t_2\rangle ...[t_n\rangle M_n \equiv \bar{M}\}.\end{aligned}$$

For ordinary nets, we will suppress W in the notation.

Definition 0.3 *For $\mathcal{N} = (S, T, F, W)$, set $<:= F^+$ and $\leq:= F^*$. For $x \in S \cup T$, set*

$$Fx := \{y : \ (y,x) \in F\},\ xF := \{y : (x,y) \in F\},\ FxF := Fx \cup xF.$$

The **conflict** *relation $\#$ is given by: $x\#y$ iff there exist $s \in S$ and $t_1, t_2 \in sF$ such that $t_1 \neq t_2$, $t_1 \leq x$, and $t_2 \leq y$.*

The interpretation justifying the notion of *conflict* will be given below for a more specialized context.

Definition 0.4 *An ordinary petri net $\mathcal{N} = (B, E, F)$ is an* **occurrence net** *(*ON *) iff:*

1. **no backward branching**: $|Fb| \leq 1$ *for all $b \in B$;*
2. **Acyclicity**: $\neg(x < x)$ *for all $x \in B \cup E$; and*
3. **absence of self-conflict**: $\neg(x\#x)$ *for all $x \in B \cup E$.*

If also $|bF| \leq 1$ for all $b \in B$, $\mathcal{N}$ is called a **causal net** *or* CN. *For an* ON $\mathcal{N}$, *set*

1. *x li y iff $x < y$ or $y < x$, and*
2. *x co y iff $x \neq y$ and neither x li y.*

Denote the set of maximal *co*−cliques by $\mathcal{CUTS}(\mathcal{N})$ and set
$\mathcal{SCUTS}(\mathcal{N}) := \mathcal{CUTS}(\mathcal{N}) \cap \mathcal{P}(S)$. – The following properties are easily verified:

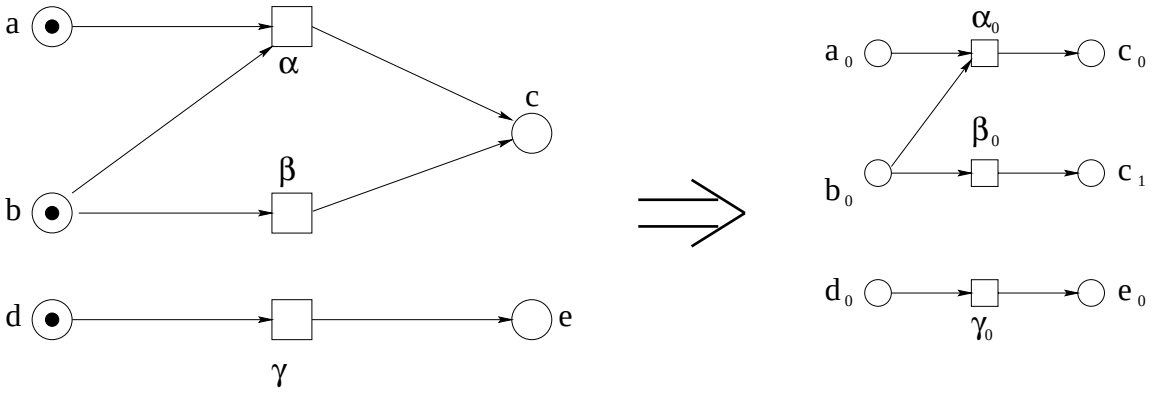


Figure 1: A 1-safe system with BP in the Engelfriet sense

Lemma 0.5 *If $\mathcal{N} = (B, E, F)$ is an* ON*, the following holds:*

1. *$<$ is a partial order (i.e. irreflexive and transitive);*
2. *li, co, and # are symmetric and irreflexive;*
3. *with $id := \{(x, x) : x \in B \cup E\}$, the product set $(B \cup E) \times (B \cup E)$ is the disjoint union of id, li, co, and #.*

Moreover, $\mathcal{N}$ is a CN *iff # is empty .*

□

Informally speaking, Branching processes (BPs) are unfoldings of Petri net systems into occurrence nets obtained from the firing rule.

Engelfriet's [2] definition of BPs requires that $\mathcal{N}$ is ordinary, and Σ is 1-safe: An Engelfriet BP of Σ is a pair $(\bar{\mathcal{N}}, p)$, where

1. $\bar{\mathcal{N}} = (B, E, \bar{F})$ is an ON such that $min(\bar{\mathcal{N}}) \in \mathcal{SCUTS}$,
2. $p : B \cup E \to S \cup T$ a labeling function such that
   (a) $p(B) \subset S$ and $p(E) \subset T$,
   (b) for all $e \in E$, p induces an isomorphism between the subnets spanned by $\bar{F}e\bar{F}$ and FtF, where $p(e) = t$.
   (c) $\forall\, e_1, e_2 \in E : (\bar{F}e_1 = \bar{F}e_2 \wedge p(e_1) = p(e_2)) \Rightarrow e_1 = e_2$, and
   (d) $p_{|_{min(\mathcal{N})}}$ is a bijection from $min(\bar{\mathcal{N}})$ to m_0.

Figure 1 illustrates the construction of BPs.

We generalize that definition to include general markings and general arc weights:

Definition 0.6 *Let $\Sigma = (\mathcal{N}, M_0)$ be a system with $\mathcal{N} = (S, T, F, W)$. A* **(generalized) branching process** *of Σ is a triple $\Pi = (\bar{\mathcal{N}}, p, l)$, where $\bar{\mathcal{N}} = (B, E, \bar{F})$ is an* ON *with $min(\bar{\mathcal{N}}) \in \mathcal{SCUTS}(\bar{\mathcal{N}})$, and*

$$p : (B \cup E) \to (S \cup T) \quad , \quad l : B \to S$$

are mappings such that

1. $p(B) \subset S$ *and* $p(E) \subset T$,
2. $\forall\ e_1, e_2 \in E : (\bar{F}e_1 = \bar{F}e_2 \wedge p(e_1) = p(e_2)) \Rightarrow e_1 = e_2$,
3. $p_{|_{min(\bar{\mathcal{N}})}}$ *is a bijection between* $min(\bar{\mathcal{N}})$ *and* S,
4. $\forall\ b \in min(\bar{\mathcal{N}}) : l(b) = M_0(p(b))$,
5. $\forall\ e \in E$: *with* $p(e) = t$,
   (a) *for any* $s \in FtF$ *there exists a unique* $(b_1, b_2) \in [\bar{F}e \times e\bar{F}]$ *such that* $p(b_1) = p(b_2) = s$.
   (b) *For all* $e \in E$ *and* b_1, b_2 *according to part 5a,*
      i. $l(b_1) \geq W(p(b_1), p(e))$ *and*
      ii. $l(b_2) = l(b_1) - W(p(b_1), p(e)) + W((p(e), p(b_2)))$.
6. $\forall\ b_1, b_2 \in B : b_1\ co\ b_2 \Rightarrow p(b_1) \neq p(b_2)$.

So in our definition, every transition occurring is reflected in such a way that its pre- and its post-domain are jointly mirrored by corresponding conditions both in the pre- and the post domain of the corresponding event in the BP. The *structural* mapping p encodes the two-fold effect of any transition, forward and backward, whereas l encodes the *marking behavior*. In particular, the BP semantics are generated by the *quantitative* change of markings in $\mathcal{N}$; the 'identity' of a given token, the history of the way past conflicts have been resolved, is ignored by p and l, as it is ignored by the firing rule of Petri nets.

For BPs, the relations from Definition 0.4 have a *meaning* in terms of local aspects of the system behavior. $x\ li\ y$ means that the events (or place markings) represented by x and y are causally ordered. For $x\#y$, both are incompatible, i. e. reaching x entails never reaching y in past, present, or future (although equivalent elements may be attainable), and $x\ co\ y$ concurrency (in a narrow sense). As an example,

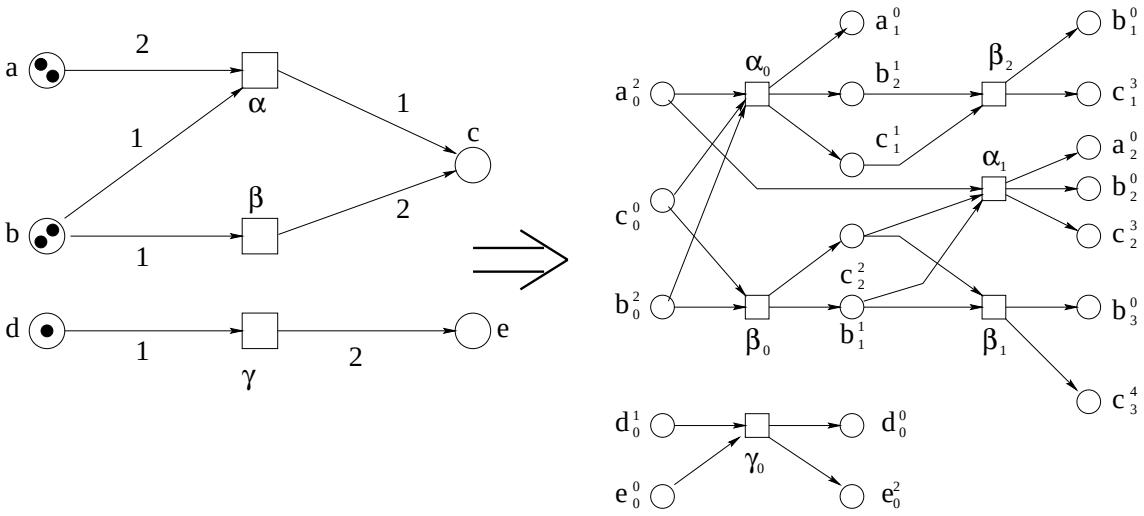


Figure 2: Maximal BP of a system with arc weights

consider figure 2. The final markings reachable are M_1 and M_2, where

$$M_1(a) = M_1(b) = M_1(d) = 0,\ M_1(c) = 3,\ M_1(e) = 2$$
$$\textit{and}\quad M_2(a) = 2,\ M_2(b) = M_2(d) = 0, M_2(c) = 4,\ M_2(e) = 2 \quad ;$$

this corresponds to the cuts

$$\gamma_1 = \{a^0_1, b^0_1, c^3_1, d^0_0, e^2_0\} \quad \textit{and} \quad \gamma_2 = \{a^2_0, b^0_2, c^4_2, d^0_0, e^2_0\}$$

in the branching process. – In fact, the BP here is *maximal*.

Let us turn towards some other aspects of the semantics. In figure 2, transitions α and β are 'in some way' in conflict with one another, depending on the situation; in the initial marking depicted, they may fire in parallel, but after one of them has fired, α and β 'battle' over the remaining token on b. The *conflict* situation is, as expected, represented by conflict in the BP. Now, the *parallelism* is reflected in the BP by the presence of all orderings of the corresponding events in, respectively, those branches of the process in which both occur.

On the other hand, true concurrency – by which we mean the *absence* of ordering between events – arises iff the extensions are disjoint: consider β and γ. All pairs of their occurrences are co- pairs in the BP; this is the case – and is only possible – for any pair t_1, t_2 of transitions if $Ft_1F \cap Ft_2F = \emptyset$. So we have, all in all, four different ways in which two occurrences e_1 and e_2 of transitions t_1, t_2 can be related:

1. causal ordering, reflected by li ,
2. conflict (#),

3. parallelism: $Ft_1F \cap Ft_2F \neq \emptyset$, but the common upstream places contain enough tokens to allow both t_1 and t_2 to fire; in this case, e_1 will occur before e_2 in some of the branches in the BP and order reversed in others, and finally

4. concurrency: $Ft_1F \cap Ft_2F = \emptyset$, and hence e_1 *co* e_2.

Note that case 3 can arise even 'between' a transition and itself (i.e. $t_1 = t_2$) whereas, obviously, case 4 cannot.

First, some general structural results for ONs.

Theorem 0.7 *Let $\bar{\mathcal{N}} = (B, E, \bar{F})$ be an* ON.

1. *For any $e \in E$ and $x \in [(B \cup E) - \{e\}]$ such that x co e, x co b for any $b \in \bar{F}e\bar{F}$.*

2. *For any $e \in E$ and $x \in [(B \cup E) - \{e\}]$ such that x co b for any $b \in \bar{F}e$, x co e.*

3. *For any $e \in E$ and $x \in [(B \cup E) - \{e\}]$ such that x co b for any $b \in e\bar{F}$, x co e.*

4. *For all $c \in \mathcal{SCUTS}(\mathcal{N})$ and $e \in E$ such that $\bar{F}e \subset c$, set $c' := [c - \bar{F}e] \cup e\bar{F}$. Then $c \in \mathcal{SCUTS}(\mathcal{N})$.*

Proof:

Proof of 1: First, take $b \in \bar{F}e$ such that $\neg(x \; co \; b)$. Then we have to consider the following cases:

1. $x < b$: Then also $x \leq e$ contradicting the assumption.
2. $b < x$: Then either $e \leq x$ or $e\#x$, again a contradiction.
3. $b\#x$: Then there exist $b \in B$ and $e_1, e_2 \in b\bar{F}$ such that $e_1 \neq e_2$, $e_1 < b$, and $e_2 \leq x$. But then $e_1 < e$ and hence $e\#x$.

Thus we have contradictions in all cases. For $b \in e\bar{F}$, the proof is analogous. □

Proof of 2: Assume $\neg(x \; co \; e)$. Once again, we have three cases all of which lead to contradictions:

1. $x < e$ implies the existence of $b \in \bar{F}e$ such that $x \leq b$.
2. $e < x$ implies $b < x$ for all $b \in \bar{F}e$.
3. $e\#x$ implies the existence of $b \in B$ and $e_1, e_2 \in b\bar{F}$ such that $e_1 \neq e_2$ and $e_1 \leq e$, $e_2 \leq x$. But then there exists $b \in \bar{F}e$ such that $x\#b$.

□

Proof of 3: Assume $\neg(x\ co\ e)$. Again, we have three cases:

1. $x < e$ implies $x < b$ for all $b \in e\bar{F}$.
2. $e < x$ implies the existence of $b \in e\bar{F}$ such that $x \leq b$.
3. $e\#x$ implies the existence of $b \in B$ and $e_1, e_2 \in b\bar{F}$ such that $e_1 \neq e_2$ and $e_1 \leq e$, $e_2 \leq x$. But then $x\#b$ for all $b \in e\bar{F}$.

□

Proof of 4: a consequence of parts 2 and 1. □

□

If c' arises from c as described in the proof, we write $c[[e\rangle\rangle c'$.

Definition 0.8 *Let $\Pi_1 = (\bar{\mathcal{N}}_1, p_1, l_1)$ and $\Pi_2 = (\bar{\mathcal{N}}_2, p_2, l_2)$ be two processes of Σ. Then: $\Pi_1 \prec \Pi_2$ iff $\bar{\mathcal{N}}_1$ is isomorphic to an initial segment of $\bar{\mathcal{N}}$.*

It can be shown that $\prec$ is a partial order; the most important result concerning $\prec$ is

Theorem 0.9 *For every system Σ, there is – up to isomorphism – a unique $\prec$ maximal process $\Pi = (\bar{\mathcal{N}}, p, l)$ of Σ.*

Proof: Analogous to Engelfriet [2]. □

By induction, one finds that BPsdo in fact reflect the behavior of Σ:

Theorem 0.10 *Let $\Sigma = (\mathcal{N}, M_0)$ with $\mathcal{N} = (S, T, F, W)$ and $\Pi = (\bar{\mathcal{N}}, p, l)$, the maximal process of Σ, with $\bar{\mathcal{N}} = (B, E, \bar{F})$. Then for every $n \geq 1$ and every firing sequence $M_0[t_0\rangle M_1[... >\rangle_n$, there exist $c_0, ..., c_n \in \mathcal{SCUTS}(\bar{\mathcal{N}})$ and $e_0, ..., e_n \in E$ such that*

1. *$c_0 = min(\bar{\mathcal{N}})$ and $c_0[[e_0\rangle\rangle c_1[[..\rangle\rangle c_n$*
2. *for all $i = 1, ..., n$ and all $b \in c_i$, $M_i(p(b)) = l(b)$.*

□

One therefore has a correspondence from markings in the system to S-cuts of the BP net in the sense that markings reachable in Σ can be 'reached' in $\bar{\mathcal{N}}$; in fact, $[[\rangle\rangle$ defines a 'pseudo-firing' in the unfolded net that simulates the firings in Σ. If $\mathcal{N}$ is finite[1], the correspondence is actually two-way, i.e. for every $c \in \mathcal{SCUTS}(\bar{\mathcal{N}})$

[1] in fact, the claim made subsequently is valid under more general assumptions; but finiteness is already general enough to cover all practical cases

there is a *reachable* marking M_c represented by c. Generalized BPs exist for general net systems with arc weights and arbitrary markings. They provide an ON semantics with two particular properties: it represents both the upstream and the downstream effects of any given transition, and it discriminates different kinds of marking-dependent interrelations between occurrences. For finite nets, they can be inductively defined; there always exists, as in the 1-safe case, a maximal BP .

Definition 0.6 extends Engelfriet's definition of branching processes; it is at the same time the 'branching version' of the *executions* as defined and studied by Vogler [8].

The construction of a generalized BP can – for bounded nets – equivalently be obtained in the following way: transform the original system into an equivalent 1-safe system by introducing n_p places for every place p, where n_p is the maximal number of tokens on p, and the corresponding marking and transitions; then, generate the Engelfriet BP of the new system. This construction – which is mentioned by Baumgarten [1] – requires a lot more space and time than the direct one we propose; for constructing a generalized BP , one needs only to introduce a new condition (standing for a number of tokens on some place) when necessary. Since in general not all amounts of tokens between 0 and n_p will be realized but only a few of them, Definition 0.6 helps reduce overhead.

Turning towards applications, it is already known that one can obtain a suitable finite prefix of a BP by stopping the unfolding after the occurrence of so called 'cut-off events'. These were introduced by McMillan [6] and used in the PEPtool's model checker [3]; cf. Graves [4] for the appropriate adjustment of the definition. Their approach carries over to our BPs, thus model checking of more general logics and without assumptions on safeness becomes possible.

References

[1] B. Baumgarten. *Petri-Netze. Grundlagen und Anwendungen.* BI, Mannheim / Wien / Zürich, 1990.

[2] J. Engelfriet. Branching Processes of Petri Nets. *Acta Informatica*, 28:575–591, 1991.

[3] J. Esparza. Model Checking Using Net Unfoldings. *Science of Computer Programming*, 23:151–195, 1994.

[4] B. Graves. Computing Reachability Properties Hidden in Finite Net Unfoldings. In *Proceedings of 17th FSTTCS*, volume 1346 of *LNCS*, pages 327–341, Berlin, 1997. Springer.

[5] J. Esparza and S. Römer and W. Vogler. An Improvement of McMillan's Unfolding Algorithm. In T. Margaria and B. Steffen, editor, *Proceedings of TACAS '96*, volume 1055 of *LNCS*, pages 87–106, Berlin etc., 1996. Springer.

[6] G. McMillan. Using Unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In *4th Workshop on Computer Aided. Verification, Montreal 1992*, pages 164–174, 1992.

[7] M. Nielsen, G. Plotkin, , and G. Winskel. Petri nets, event structures, and domains. Part I. *Theoretical Computer Science*, 13:85–108, 1981.

[8] W. Vogler. Executions: A New Partial Order Semantics of Petri Nets. *TCS*, 91:205–238, 1991.

Automatically Proving Up-to Bisimulation

Daniel Hirschkoff
CERMICS - ENPC/INRIA
dh@cermics.enpc.fr

Abstract

We present a tool for checking bisimilarities between π-calculus processes with the *up-to techniques* for bisimulation. These techniques are used to reduce the size of the relation one has to exhibit to prove a bisimulation. Not only is this interesting in terms of space management, but it also increases dramatically the expressive power of our system, by making in some cases the verification of infinite states space processes possible. Based on an algorithm to compute a unique normal form for structural congruence, we develop sound and complete methods to check bisimulation up to injective substitutions on free names, up to restriction and up to parallel composition. We show the expressiveness of our techniques on a prototype implementation.

Introduction

We present a tool for automatically checking bisimilarities between π-calculus processes with the up-to techniques for bisimulation. Existing tools for automatically checking bisimilarities between CCS or π-calculus processes can handle a restricted class of processes that have an infinite behaviour. Methods like the partition refinement algorithm ([PT87], used for example in [PS96]) are based on a preliminary step in which the unfolding of the processes is computed, under the form of a *Labelled Transition System*. Therefore, processes having an infinite state space cannot be taken into consideration by this approach. The *Mobility Workbench* [VM94] uses an "on the fly" method, that progressively builds the candidate bisimulation relation as new pairs of related processes are discovered. This way, one can also take into account two non terminating processes that show a different behaviour after a finite number of steps, and prove that they are not bisimilar. However, two processes having an infinite states space still cannot be proven bisimilar.

In this paper, a different approach, based on the so called up-to proof techniques, is investigated, in order to define some methods for checking bisimilarities between processes. These techniques have been introduced as meta-level tools for proving bisimulation relations [SM92, San95], and to our knowledge have only been used in papers about the theory of π-calculus, to prove bisimilarity laws. They allow one to perform some syntactical manipulations on processes in order to reduce the size of the relations one has to exhibit to

prove bisimulation. The aim of this work is to investigate to what extent these techniques can be mechanised, and how such theoretical tools can be used in the context of verification. The main benefit we get from these techniques arises at the level of expressiveness: we can indeed prove in some cases bisimilarity between replicated processes having an infinite states space.

The basis for the development of our up to checking methods is the up to structural congruence proof technique. To work up to structural congruence means to have a notion of unique normal form for this equivalence. We achieve this in a simple way through the definition of a term rewriting system on a small but expressive language. This allows us to work only with terms in normal form, and to introduce effective characterisations of the various proof techniques we study (up to injective substitutions on names, up to restriction, and up to parallel composition). We rely on these characterisations to define our bisimilarity checking algorithm, which is a straightforward extension of the "on the fly" checking method for bisimulation.

For the sake of brevity, the presentation of the definitions and of the main results has been kept rather succinct (the reader should refer to [Hir98] for the proofs of the properties we state and for more comments on the design choices). Insight on the technical issues that arise as we mechanise the up to techniques is given along the statements of our results. The paper is organised as follows: Section 1 describes the general framework of our study: after defining the syntax of our terms and structural congruence, we introduce a normalisation algorithm that enjoys the uniqueness of normal forms property. We then introduce semantics and the behavioural equivalence we use on processes, namely bisimilarity, and give a brief account on the up to proof techniques for bisimulation. In Section 2, we define the up to techniques we use, and give characterisations of the corresponding functions on relations. We put together these results in Section 3 to build a prototype implementation, and illustrate the behaviour of our techniques on a few simple examples. We finally conclude with a brief discussion on the insight brought by our study, and on future work.

1 Definitions and Notations

1.1 Syntax

Let $a, b, \ldots, x, y, \ldots$ range over an infinite countable set of *names*, and $\vec{a}, \vec{b}, \ldots$ range over (possibly empty) name lists. Processes, ranged over by $P, Q, \ldots$, are defined by the following syntax:

$$\alpha \;=\; a(\vec{b}) \mid \overline{a}[\vec{b}]\,, \qquad\qquad P \;=\; \mathbf{0} \mid \alpha.P \mid\; !\alpha.P \mid (\nu x)\,P \mid P_1|P_2\,.$$

Prefixes are either input: $a(\vec{b})$, or output: $\overline{a}[\vec{b}]$. $\mathbf{0}$ is the inactive process; prefixed processes are either linear ($\alpha.P$) or replicated ($!\alpha.P$);the other constructors are restriction (ν) and parallel composition ($|$). Bound names are defined by saying that restriction and input prefix are binding operators: (νx) and $x(\vec{y})$ respectively bind name x and the names

in $\vec{y}$ in their continuation. As usual, free names are names that are not bound in a process, and we work up to implicit α-conversion of bound names (at least until Section 2, where α-conversion will be handled explicitly for the definition of our checking methods).

Structural congruence, written $\equiv$, is the smallest equivalence relation that is a congruence and that satisfies the following rules:

$$1\ \ P|\mathbf{0} \equiv P \qquad 2\ \ P|Q \equiv Q|P \qquad 3\ \ P|(Q|R) \equiv (P|Q)|R$$

$$4\ \ (\nu x)(\nu y)\,P \equiv (\nu y)(\nu x)\,P \qquad 5\ \ (\nu x)\,\mathbf{0} \equiv \mathbf{0} \qquad 6\ \ \frac{x \notin fn(P)}{P|(\nu x)\,Q \equiv (\nu x)\,(P|Q)}$$

$$7\ \ !\alpha.P|\alpha.P \equiv !\alpha.P \qquad 8\ \ !\alpha.P|!\alpha.P \equiv !\alpha.P$$

Rules 1-3 give properties about the parallel composition operator, rules 4-6 deal with restriction, and rules 7-8 with replication. Rule 8 is new with respect to the traditional definition of structural congruence [Mil91], and can be seen as the "limit" of infinitely many applications of Rule 7.

Conventions and notations: Rules 2 and 3 (for parallel composition) and 4 (for restriction) will be used implicitly, which means that we work up to commutativity and associativity of parallel composition, and up to permutation of consecutive restrictions. This will allow us to use the notation $(P_1|\ldots|P_n)$ (sometimes abbreviated as $\prod_{i\in[1,\ldots,n]} P_i$) for parallel composition, and $(\nu\vec{x})\,P$ for restriction, where $\vec{x}$ is intended as having a set rather than a vector structure (in order to allow silent applications of rule 4). The technical issues raised by the implementation of these aspects of structural congruence will be discussed in Section 2.

Whereas rules 1 and 5 are used to do some garbage collection, the rules that will be really relevant in the definition of our notion of normal form are rules 6, 7 and 8, as will be seen below. Remark that structural congruence preserves free names, i.e if $P \equiv Q$, then $fn(P) = fn(Q)$.

Notation: In the following, we write $=_{\alpha\nu}$ to denote equality between processes up to α-conversion, permutation of consecutive restrictions (structural congruence law 4), and commutativity and associativity of parallel composition. With this notation, we focus on the interplay between α-conversion and permutation of consecutive restrictions, leaving the management of parallel compositions aside, as this is a somewhat orthogonal question.

1.2 Normal Forms

We give an orientation to the relevant structural congruence laws to define a term rewriting system as follows:

Definition 1.1 (Normalisation algorithm) *The normalisation algorithm is defined as*

the rewriting system given by the five following rules[1]*:*

$$R5 \quad (\nu x)\,\mathbf{0} \to \mathbf{0} \qquad R6 \quad (x \notin fn(P)) \Rightarrow (P|(\nu x)\,Q \to\ (\nu x)\,(P|Q))$$

$$R1 \quad P|\mathbf{0} \to\ P \qquad R7 \quad !\alpha.P|\alpha.P \to !\alpha.P \qquad R8 \quad !\alpha.P|!\alpha.P \to !\alpha.P$$

This rewriting system enjoys strong normalisation (a computation always terminates) and local confluence (two one-step reducts of a term can always be rewritten into a common term), which guarantees uniqueness of normal forms:

Proposition 1.2 (Uniqueness of normal forms) *For any process P, there exists a unique process, written $\mathbf{NR}(P)$, obtained by application of our rewriting system to P , and that cannot be further rewritten. Moreover, given two processes P and Q, $P \equiv Q$ if and only if $\mathbf{NR}(P) =_{\alpha\nu} \mathbf{NR}(Q)$.*

We now give a syntactical description of the terms that cannot be rewritten by our algorithm:

For $m \in \mathcal{N}$, define $(\alpha.N)^m = \overbrace{\alpha.N|\ldots|\alpha.N}^{m \text{ times}}$ and let $(\alpha.N)^\omega = !\alpha.N$.

$$\begin{array}{rcl} N & = & \mathbf{0} \\ & | & (\nu\vec{x})\,((\alpha_1.N_1)^{m_1}|\ldots|(\alpha_n.N_n)^{m_n})\,, \qquad n \geq 1,\ m_i \in \mathcal{N} \cup \{\omega\} \\ & & \left\{ \begin{array}{l} \forall i.\, x_i \in fn((\alpha_1.N_1)^{m_1}|\ldots|(\alpha_n.N_n)^{m_n}) \\ \forall i,j \in [1,\ldots,n].\,(i \neq j) \Rightarrow (\alpha_i.N_i \neq \alpha_j.N_j) \end{array} \right. \end{array}$$

Figure 1: Syntax of normal forms

Proposition 1.3 (Syntactical description of normal forms) *The terms that are of the form $\mathbf{NR}(P)$, for some P, are exactly those described by the syntax defined in Figure 1.*

Let us comment on the shape of normal forms: the syntax of Figure 1 says that every process that is not equivalent to $\mathbf{0}$ can be viewed as an agent with two components, its body and its topmost restrictions. The body is made of processes that are ready to commit, and the topmost restrictions define some kind of geometry among these processes.

Notations: Given a non-null process in normal form $P = \prod_i (\alpha_i.N_i)^{m_i}$, we write $P = (\nu\vec{x}_P)\,\langle P\rangle$ to decompose P into its uppermost restrictions $(\nu\vec{x}_P)$ and its "body" $\prod_i (\alpha_i.N_i)^{m_i}$, which consists of (possibly replicated) prefixed processes. Note that bodies of normal forms are also normal forms. We will range over such processes (i.e. non-null normal forms without topmost restrictions) with the notation $\langle P\rangle, \langle Q\rangle, \ldots$. We further decompose $\langle P\rangle$ into an "infinite part", written $\langle P\rangle_\omega$, and a "finite part", written

[1]Note that rule R6 is guarded by a condition; however, we can consider our system as a usual Term Rewriting System (i.e. without conditions), for example by adopting a De Bruijn notation for names, which intrinsically embeds the side condition when applying the structural congruence rule.

$\langle P\rangle_{\mathcal{N}}$, respectively corresponding to the replicated and the non-replicated components, i.e. $\langle P\rangle_\omega = \prod_{i.\, m_i=\omega}(\alpha_i.N_i)^{m_i}$ and $\langle P\rangle_{\mathcal{N}} = \prod_{i.\, m_i\in\mathcal{N}}(\alpha_i.N_i)^{m_i}$.

We introduce some machinery on processes of the form $\langle P\rangle$, that will be useful for the treatment of the up to parallel composition proof technique in Section 2: we let

$$\prod_i(\alpha_i.P_i)^{m_i} \setminus \prod_j(\beta_j.Q_j)^\omega \stackrel{def}{=} \prod_{i.\forall j.\,\alpha_i.P_i\neq\beta_j.Q_j}(\alpha_i.P_i)^{m_i}$$

(note that the right hand side argument of $\setminus$ is always of the form $\langle P\rangle_\omega$), and, for two processes of the form $\langle P\rangle$ and $\langle Q\rangle$, we let $\langle P\rangle\oplus\langle Q\rangle \stackrel{def}{=} \langle P\rangle_\omega \mid (\langle Q\rangle_\omega \setminus \langle P\rangle_\omega) \mid (\langle P\rangle_{\mathcal{N}} \setminus \langle Q\rangle_\omega) \mid (\langle Q\rangle_{\mathcal{N}} \setminus \langle P\rangle_\omega)$.

1.3 Semantics

1.3.1 Operational Semantics and Bisimulation

$$INP\quad a(\vec{x}).P \xrightarrow{a(\vec{b})} P_{\{\vec{x}:=\vec{b}\}} \qquad RES\quad \frac{P \xrightarrow{\mu} P'}{(\nu x)\,P \xrightarrow{\mu} (\nu x)\,P'}\quad x\notin n(\mu)$$

$$OUT\quad \overline{a}[\vec{b}].P \xrightarrow{\overline{a}[\vec{b}]} P \qquad OPEN\quad \frac{P \xrightarrow{(\nu\vec{b'})\,\overline{a}[\vec{b}]} P'}{(\nu x)\,P \xrightarrow{(\nu t::\vec{b'})\,\overline{a}[\vec{b}]} P'}\quad \begin{cases} x\neq a \\ x\in\vec{b}\setminus\vec{b'} \end{cases}$$

$$BANG\quad \frac{\alpha.P \xrightarrow{\mu} P'}{!\alpha.P \xrightarrow{\mu} !\alpha.P|P'} \qquad PAR_l\quad \frac{P \xrightarrow{\mu} P'}{Q|P \xrightarrow{\mu} Q|P'}\quad fn(Q)\cap bn(\mu)=\emptyset$$

$$CLOSE_1\quad \frac{P \xrightarrow{a(\vec{b})} P' \qquad Q \xrightarrow{(\nu\vec{b'})\,\overline{a}[\vec{b}]} Q'}{P|Q \xrightarrow{\tau} (\nu\vec{b'})\,(P'|Q')}$$

Figure 2: Early Transition Semantics

The rules for *early* transition semantics are given in Figure 2 (:: denotes the adjunction of an element to a list; symmetrical versions of rules PAR_l and $CLOSE_1$ have been omitted; note the particular shape of rule $BANG$, in relation to our syntax). The semantical equivalence on processes we use is bisimilarity, defined as follows:

Definition 1.4 (Bisimulation, bisimilarity) *A relation $\mathcal{R}$ is a* bisimulation *iff for every pair of processes (P,Q) such that $P\mathcal{R}Q$, whenever $P \xrightarrow{\mu} P'$, there exists a process Q' such that $Q \xrightarrow{\mu} Q'$ and $P'\mathcal{R}Q'$, and the symmetrical condition on transitions performed by Q. Bisimilarity, written $\sim$, is the greatest bisimulation.*

1.3.2 The Up-to Proof Techniques for Bisimulation

To rephrase Definition 1.4 above, proving bisimilarity of two processes reduces to exhibiting a bisimulation relation that contains these processes. The property "to be a bisimulation relation" can be depicted by the diagram on the left side of Figure 3: $\mathcal{R}$ is a bisimulation if any pair of processes in $\mathcal{R}$ evolves to pairs of processes that are also in $\mathcal{R}$. In other words, $\mathcal{R}$ contains the whole "future" of all the processes it relates.

Figure 3: From bisimulation to up-to bisimulation

In [San95], Sangiorgi introduces a general framework for the study of the *up-to techniques*, which can be used to reduce the size of the relations one has to exhibit in order to prove bisimulation. Each technique is represented by a functional from relations to relations (ranged over by $\mathcal{F}$):

Definition 1.5 (Bisimulation up to $\mathcal{F}$) *Given a functional $\mathcal{F}$ over relations, we say that a relation $\mathcal{R}$ is a bisimulation up to $\mathcal{F}$ iff, for every P and Q such that $P\mathcal{R}Q$, whenever $P \xrightarrow{\mu} P'$, there exists Q' s.t. $Q \xrightarrow{\mu} Q'$ and $P'\mathcal{F}(\mathcal{R})Q'$, and the symmetrical condition on transitions performed by Q.*

A functional $\mathcal{F}$ gives a correct proof technique if it is *sound*, i.e. if ($\mathcal{R}$ is a bisimulation up to $\mathcal{F}$) implies ($\mathcal{R} \subseteq \sim$), which means that in some way, $\mathcal{F}$ helps building the "future" of a relation: to prove that $\mathcal{R}$ is a bisimulation relation, it is enough to prove that any pair of processes in $\mathcal{R}$ can only evolve to pairs of processes that are contained in $\mathcal{F}(\mathcal{R})$ (as shown on the right part of Figure 3). [San95] introduces a sufficient condition for soundness of functionals, called *respectfulness*. All the techniques we are using in the remainder of the paper (up to injective substitutions on free names, up to structural congruence, up to restrictions, up to parallel composition) are respectful, and can be combined together for the task of proving bisimulations, thanks to nice compositionality properties of respectful functions.

2 Automatising the Up-to Techniques

In this Section, we define sound and complete methods to decide, given a relation $\mathcal{R}$, if a pair of processes belongs to $\mathcal{F}(\mathcal{R})$, for some function $\mathcal{F}$ corresponding to a correct proof technique. Using Definition 1.5, this amounts to define a checking method to tell if a relation is a bisimulation up to $\mathcal{F}$.

The normalising function we have defined gives rise to the bisimulation up to bisimilarity proof technique. We now study the up to injective substitutions on free names, the up to restriction and the up to parallel composition proof techniques. The corresponding functions are proved bisimilar in [San95]; however, their nice compositionality properties do not extend to the characterisations, which prevents us to treat them separately. Therefore, we shall treat them incrementally, adding a technique at each step, and leading to our most powerful technique in Definition 2.7. Notice anyway that the overall methodology of our checking methods is rather uniform. Because of lack of space, we will just state the definitions and characterisations of the various proof techniques (the reader interested in

a more detailed discussion and in the proofs should refer to [Hir98]), and then comment about the algorithms induced by our characterisations.

2.1 Definitions and Characterisations

We first need some background on substitutions on names, that are functions from names to names, ranged over by σ, σ', σ''. We define $dom(\sigma)$, the domain of σ, as the set of names n such that $\sigma(n) \neq n$, and the codomain $codom(\sigma)$ of σ as $\sigma(dom(\sigma))$. σ is *injective* if $\sigma(i) = \sigma(j)$ implies $i = j$. In the following, we are interested, given a process P, in substitutions σ that are injective on the free names of P, and such that applying σ to P does not capture bound names of P, i.e. $codom(\sigma) \cap bn(P) = \emptyset$ (this is always possible modulo α-conversion). An injective substitution σ whose domain is finite defines a bijective mapping between $dom(\sigma)$ and $codom(\sigma)$; we shall write σ^{-1} for the inverse of σ. Given a set of names E, we say that two substitutions σ and σ' *coincide* on E, written $\sigma = \sigma'$ on E, iff for any name n in E, $\sigma(n) = \sigma'(n)$.

2.1.1 Up to Injective Substitutions on Names

Definition 2.1 *Given a relation $\mathcal{R}$, we define the* closure under structural congruence and injective substitutions on free names *of $\mathcal{R}$, written $\equiv\mathcal{R}^i\equiv$, as follows:*

$$\equiv\mathcal{R}^i\equiv \stackrel{def}{=} \{(P,Q); \exists (P_0,Q_0) \in \mathcal{R}, \exists \sigma \text{ injective on } fn(P_0)\cup fn(Q_0). \\ P \equiv P_0\sigma \wedge Q \equiv Q_0\sigma\}.$$

In order to give a characterisation of $\equiv\mathcal{R}^i\equiv$, we study α-convertibility of processes in normal form, and its relation to injective substitutions:

Fact 2.2 *Given two processes P and Q in normal form, write $P = (\nu\vec{x})\langle P\rangle$ and $Q = (\nu\vec{y})\langle Q\rangle$. Then $(\nu\vec{x})\langle P\rangle =_{\alpha\nu} (\nu\vec{y})\langle Q\rangle$ iff there exists a substitution σ, injective on $fn(\langle Q\rangle)$, s.t. $\langle P\rangle =_{\alpha\nu} \langle Q\rangle\sigma$, $dom(\sigma) = \vec{y}$ and $\sigma(\vec{y}) = \vec{x}$.*

Lemma 2.3 *Write as above $P = (\nu\vec{x})\langle P\rangle$ and $Q = (\nu\vec{y})\langle Q\rangle$, for two normal processes P and Q. Then, for any substitution σ injective on the free names of $\langle Q\rangle$, $(\nu\vec{x})\langle P\rangle =_{\alpha\nu} (\nu\vec{y})(\langle Q\rangle\sigma)$ iff there exists a substitution σ' injective on $fn(\langle Q\rangle)$ s.t. (i) $\langle P\rangle =_{\alpha\nu} \langle Q\rangle\sigma'$, (ii) $\sigma'(\sigma^{-1}(\vec{y})) = \vec{x}$, and (iii) $\sigma' = \sigma$ on $fn(\langle Q\rangle) \setminus \sigma^{-1}(\vec{y})$.*

Proposition 2.4 (Characterisation of $\equiv\mathcal{R}^i\equiv$) *$(P,Q) \in \equiv\mathcal{R}^i\equiv$ iff there exist processes P_0, Q_0, and substitutions σ' σ'', injective on $fn(\langle P_0\rangle)$ and $fn(\langle Q_0\rangle)$ respectively, such that, if we write $P_0 = (\nu\vec{x}_{P_0})\langle P_0\rangle$, $Q_0 = (\nu\vec{x}_{Q_0})\langle Q_0\rangle$, $\mathbf{NR}(P) = (\nu\vec{x}_P)\langle \mathbf{NR}(P)\rangle$ and $\mathbf{NR}(Q) = (\nu\vec{x}_Q)\langle \mathbf{NR}(Q)\rangle$, we have $P_0\mathcal{R}Q_0$, $\langle \mathbf{NR}(P)\rangle =_{\alpha\nu} \langle P_0\rangle\sigma'$, $\langle \mathbf{NR}(Q)\rangle =_{\alpha\nu} \langle Q_0\rangle\sigma''$, $\sigma'(\vec{x}_{P_0}) = \vec{x}_P$, $\sigma''(\vec{x}_{Q_0}) = \vec{x}_Q$, and, if we write $E = fn(\langle P_0\rangle) \cap fn(\langle Q_0\rangle) \setminus \vec{x}_{P_0}\vec{x}_{Q_0}$, $\sigma' = \sigma''$ on E.*

2.1.2 Up to Restriction

Definition 2.5 *We write $\equiv(\mathcal{R}^i)^\nu\equiv$ to denote the closure under injective substitutions, structural congruence and restrictions of a relation $\mathcal{R}$, defined as follows:*

$$\equiv(\mathcal{R}^i)^\nu\equiv \stackrel{def}{=} \{(P,Q);\exists P_0,Q_0,\exists\vec{v},\exists\sigma \text{ injective on } fn(P_0).\\ ((P_0,Q_0)\in\mathcal{R})\wedge(P\equiv(\nu\vec{v})\,(P_0\sigma))\wedge(Q\equiv(\nu\vec{v})\,(Q_0\sigma))\}\,.$$

Proposition 2.6 (Characterisation of the closure up to restrictions) *Given two processes P and Q and a relation $\mathcal{R}$, write $\mathbf{NR}(P)=(\nu\vec{x}_P)\,\langle\mathbf{NR}(P)\rangle$ and $\mathbf{NR}(Q)=(\nu\vec{x}_Q)\,\langle\mathbf{NR}(Q)\rangle$. Then $(P,Q)\in\equiv(\mathcal{R}^i)^\nu\equiv$ iff there exist two substitutions σ' and σ'' injective on $fn(\langle P_0\rangle)$ and $fn(\langle Q_0\rangle)$ respectively, processes P_0 and Q_0 and a name list $\vec{V}\subseteq fn(P_0)\cup fn(Q_0)$ such that, if we write $P_0=(\nu\vec{x}_{P_0})\,\langle P_0\rangle$, $Q_0=(\nu\vec{x}_{Q_0})\,\langle Q_0\rangle$, $\vec{V}_1=\vec{V}_{|fn(P_0))}$ and $\vec{V}_2=\vec{V}_{|fn(Q_0)}$:*

$$(i)\ \langle\mathbf{NR}(P)\rangle{=}_{\alpha\nu}\langle P_0\rangle\sigma' \ \text{ and } \ \langle\mathbf{NR}(Q)\rangle{=}_{\alpha\nu}\langle Q_0\rangle\sigma''$$
$$(ii)\ \sigma'(\vec{V}_1\vec{x}_{P_0})=\vec{x}_P \ \text{ and } \ \sigma''(\vec{V}_2\vec{x}_{Q_0})=\vec{x}_Q$$
$$(iii)\ \sigma'=\sigma'' \text{ on } E=fn(\langle P_0\rangle)\cap fn(\langle Q_0\rangle)\setminus(\vec{V}\vec{x}_{P_0}\vec{x}_{Q_0})\,.$$

The condition above can be seen as an enlargement of the up to injective substitutions case: the up-to restrictions technique compels σ' and σ'' to coincide on a smaller set E of names, or in other words more names in $fn(P_0)$ and $fn(Q_0)$ can be mapped to different names by σ' and σ''.

2.1.3 Up to Parallel Composition

To reason with both the up to restriction and the up to parallel composition proof techniques, we work with contexts that are described by the following syntax (note that the up to structural congruence proof technique allows us to adopt this simple form of contexts without loss of generality):

$$C \;=\; (\nu\vec{x})\,([\,]|T) \qquad T=\prod_i(\alpha_i.N_i)^{m_i},\ T \text{ in normal form}\,.$$

Definition 2.7 *We write $\equiv(\mathcal{R}^i)^{\mathcal{C}}\equiv$ to denote the closure under injective substitutions, structural congruence, restriction and parallel composition of a relation $\mathcal{R}$, defined as follows:*

$$\equiv(\mathcal{R}^i)^{\mathcal{C}}\equiv \stackrel{def}{=} \{(P,Q);\exists P_0,Q_0,\exists C,\exists\sigma \text{ injective on } fn(P_0).\\ ((P_0,Q_0)\in\mathcal{R})\wedge(P\equiv C[P_0\sigma])\wedge(Q\equiv C[Q_0\sigma])\}\,.$$

Proposition 2.8 (Characterisation of $\equiv(\mathcal{R}^i)^{\mathcal{C}}\equiv$) *Given a relation $\mathcal{R}$ and two processes P and Q, $(P,Q)\in\equiv(\mathcal{R}^i)^{\mathcal{C}}\equiv$ iff there exist $(P_0,Q_0)\in\mathcal{R}$, a process $\langle T\rangle$, two substitutions σ' and σ'' injective on $fn(\langle P_0\rangle\mid\langle T\rangle)$ and $fn(\langle Q_0\rangle\mid\langle T\rangle)$ respectively, and a name list $\vec{V}\subseteq fn(P_0)\cup fn(Q_0)\cup fn(\langle T\rangle)$ such that, if we write $\mathbf{NR}(P)=(\nu\vec{x}_P)\,\langle\mathbf{NR}(P)\rangle$, $P_0=$*

$(\nu\vec{x}_{P_0})\langle P_0\rangle$, $\vec{V}_1 = \vec{V}_{|fn(P_0)}$, $\langle T_1\rangle_\omega =_{\alpha\nu} \langle T\rangle_\omega \setminus \langle P_0\rangle_\omega$, $\langle T_1\rangle_{\mathcal{N}} =_{\alpha\nu} \langle T\rangle_{\mathcal{N}} \setminus \langle P_0\rangle_\omega$, *and similarly for* Q, Q_0, $\vec{x}_{Q_0}$, $\vec{V}_2$ *and* $\langle T_2\rangle$, *we have:*

$$(i)\ \begin{cases} \langle \mathbf{NR}(P)\rangle_\omega =_{\alpha\nu} (\langle P_0\rangle_\omega \mid \langle T_1\rangle_\omega)\sigma' \\ \langle \mathbf{NR}(Q)\rangle_\omega =_{\alpha\nu} (\langle Q_0\rangle_\omega \mid \langle T_2\rangle_\omega)\sigma'' \end{cases}$$

$$(i')\ \begin{cases} \langle \mathbf{NR}(P)\rangle_{\mathcal{N}} =_{\alpha\nu} (\langle P_0\rangle_{\mathcal{N}} \setminus \langle T_1\rangle_\omega \mid \langle T_1\rangle_{\mathcal{N}})\sigma' \\ \langle \mathbf{NR}(Q)\rangle_{\mathcal{N}} =_{\alpha\nu} (\langle Q_0\rangle_{\mathcal{N}} \setminus \langle T_2\rangle_\omega \mid \langle T_2\rangle_{\mathcal{N}})\sigma'' \end{cases}$$

$$(ii)\ \sigma'(\vec{V}_1\vec{x}_{P_0}) = \vec{x}_P \quad and \quad \sigma''(\vec{V}_2\vec{x}_{Q_0}) = \vec{x}_Q$$

$$(iii)\ \sigma' = \sigma''\ on\ E = fn(\langle P_0\rangle \oplus \langle T\rangle) \cap fn(\langle Q_0\rangle \oplus \langle T\rangle) \setminus (\vec{V}\vec{x}_{P_0}\vec{x}_{Q_0})\,.$$

2.2 On the induced checking methods

We now describe informally the implementation of the checking methods induced by Propositions 2.4, 2.6 and 2.8. The overall methodology is the same in each case; we illustrate it on the second proof technique. Given two processes P and Q, and a relation $\mathcal{R}$, to decide if $(P,Q) \in \equiv (\mathcal{R}^i)^\nu \equiv$:

1. compute $\mathbf{NR}(P)$ and $\mathbf{NR}(Q)$, yielding $\mathbf{NR}(P) = (\nu\vec{x}_P)\langle \mathbf{NR}(P)\rangle$ and $\mathbf{NR}(Q) = (\nu\vec{x}_Q)\langle \mathbf{NR}(Q)\rangle$;
2. pick $(P_0, Q_0) \in \mathcal{R}$, and compute as above $P_0 = (\nu\vec{x}_{P_0})\langle P_0\rangle$ and $Q_0 = (\nu\vec{x}_{Q_0})\langle Q_0\rangle$; if any of the checks below fails, go back to point 2 with another pair (P_0, Q_0) of processes;
3. use (i) to compute σ' s.t. $\langle \mathbf{NR}(P)\rangle =_{\alpha\nu} \langle P_0\rangle\sigma'$, and proceed similarly for σ'';
4. find $\vec{V}_1$ and $\vec{V}_2$ and extend σ' and σ'' so that (ii) and (iii) hold.

Steps 1 and 2 use the normalisation function of Section 1; the difficulties are concentrated in step 3, where we have to derive matchings between two bodies of processes, and infer the corresponding substitutions on free names. Indeed, in our mathematical reasoning, we treat process bodies as parallel compositions whose components can be implicitly rearranged the way we want (see above). When it comes to implementation, however, we have to front the question of the ordering of parallel components, which is not an easy task. Consider for example the case where $\langle \mathbf{NR}(P)\rangle = !x \mid !y \mid \overline{x}$ and $\langle P_0\rangle = !y \mid !x \mid \overline{x}$: here the two bodies can be matched together. While we can decide that the $\overline{x}$ component comes after the replicated input components, because we can distinguish these subterms "structurally", we have not been able to define a canonical ordering for the parallel composition $!x \mid !y$. Indeed, this order on processes should be invariant under injective substitution on free names (a proof technique that is involved in all our checking methods). Therefore, we cannot find an easy way to ordinate two processes that are equal up to some renaming, and we are compelled to introduce some combinatorics on the lists of parallel components of a process. In our example, we have to take into account the two possible orderings of $!x$ and $!y$ in order to infer the matching between $\langle \mathbf{NR}(P)\rangle$ and $\langle P_0\rangle$.

Similar problems arise in the treatment of the most powerful proof technique, corresponding to Proposition 2.8. In that case, we do not infer a simple matching to insure conditions (i) and (i'), but rather have to establish simultaneously a property of matching

and a property of *structural inclusion* (akin to [EG98]), meaning that a process is a subpart of a parallel composition (to isolate process T).

3 An Implementation

3.1 The Tool

We present a prototype implementation of the methods described in the latter Section, under the form of a tool for checking bisimulation using the up to techniques. This tool, written in O'Caml, allows the user to define a pair of processes, choose an up-to technique among those studied above, and try to prove bisimilarity using this technique. In the case where the proof succeeds, the corresponding bisimulation relation is displayed; if the processes are not bisimilar, some kind of diagnostic information is given to the user to justify the failure (and hopefully help him make another attempt). Other features, like the computation of the normal form of a process and the interactive simulation of the behaviour of a process, are also provided.

Note that the tool allows the user to check also *weak bisimilarity* (written $\approx$), that is defined by replacing $Q \xrightarrow{\mu} Q'$ with $Q \stackrel{*}{\Rightarrow} Q'$ if $\mu = \tau$, with $Q \stackrel{*}{\Rightarrow}\xrightarrow{\mu}\stackrel{*}{\Rightarrow} Q'$ if $\mu \neq \tau$, in Definition 1.4, where $\stackrel{*}{\Rightarrow}$ denotes the reflexive, transitive closure of $\xrightarrow{\tau}$. All the up-to techniques we have studied, as well as our results, extend directly to the weak case.

The algorithm To each proof technique $\mathcal{F}$ we have seen in Section 2 corresponds a decision procedure $\texttt{decide}_{\mathcal{F}}$, given by the characterisations of Propositions 2.4, 2.6 and 2.8. Our "bisimulation up-to $\mathcal{F}$" checking function $\texttt{bisim}_{\mathcal{F}}$ (defined on Figure 4) takes three arguments: a relation $\mathcal{R}$ and two processes P and Q, and returns an up-to $\mathcal{F}$ bisimulation relation extending $\mathcal{R}$ that contains (P, Q). This functions follows Definition 1.5, trying to build up an up-to bisimulation until it reaches a fixpoint. Its correctness derives from the soundness of the closure functions we apply to relations, as proved in [San95]. Of course, our algorithm is not complete, since in the case where the candidate bisimulation relation we generate keeps growing even up to the techniques we use, the program enters an infinite loop.

3.2 Examples

• $(\nu b)(!b.a(x).\overline{x} \mid !a(t).\overline{t} \mid !\overline{b}) \sim !a(x).\overline{x} \mid (\nu c)\,(!\overline{c} \mid !c)$: in the proof of this result, the normalisation algorithm erases each copy of $a(x).\overline{x}$ that is generated after a communication over b takes place in the left hand side process. We show a simple session, where the user defines the left and right processes (commands `Left` and `Right`), asks the system to print the pair of processes (command `Print`), and checks bisimilarity (command `Check`):

```
> Left (^b) (!b.a(x).x[] | !a(t).t[] | !b[] )
> Right !a(x).x[] | (^c)(!c[] | !c)
> Print
The pair is
```

To compute $\texttt{bisim}_{\mathcal{F}}(\mathcal{R}, P, Q)$:

- (parameter: $\mathcal{R}$) pick a transition $P \xrightarrow{\mu} P'$ of P, and compute $Q_\mu = \{Q'. Q \xrightarrow{\mu} Q'\}$;
  - use $\texttt{decide}_{\mathcal{F}}$ to check if any of the elements of Q_μ satisfies $P'\mathcal{F}(\mathcal{R})Q'$. If such an element can be found, loop to another transition of P, leaving $\mathcal{R}$ unchanged;
  - otherwise, pick a $Q' \in Q_\mu$ and make the recursive call $\texttt{bisim}_{\mathcal{F}}((P', Q') :: \mathcal{R}, P', Q')$; if this call succeeds, yielding $\mathcal{R}$', loop to another transition of P with $\mathcal{R}$', otherwise pick another $Q' \in Q_\mu$; if all the recursive calls to $\texttt{bisim}_{\mathcal{F}}$ fail, fail;
- proceed similarly with the transitions of Q.

Figure 4: The checking algorithm

```
((^b)(!b.a(x).x[] | !a(t).t[] | !b[]) , (!a(x).x[] | (^c)(!c[] | !c)))
> Check
Yes, size of the relation is 1
((^b)(!b.a(x).x[] | !a(t).t[] | !b[]),(^c)(!c | !a(x).x[] | !c[]));
```

The syntax for processes is rather intuitive; we use ^ for restriction and square brackets for emission. Both processes are weakly bisimilar to $!a(x).\overline{x}$; here the user uses command `Switch` to toggle the bisimilarity checking mode (from strong to weak):

```
> Print
The pair is
((^b)(!b.a(x).x[] | !a(t).t[] | !b[]) , !a(x).x[])
> Switch
Checking mode is weak, verbose mode is on.
> Check
Yes, size of the relation is 1
((^b)(!b.a(x).x[] | !a(t).t[] | !b[]),!a(x).x[]);
```

- Another law, which is a straightforward instantiation of the so-called *replication theorems*, that express the distributivity of private resources: $(\nu a)\,(!a(x).\overline{x} \mid !\overline{a}b \mid !\overline{a}c) \sim (\nu a)\,(!a(x).\overline{x} \mid !\overline{a}b) \mid (\nu a)\,(!a(x).\overline{x} \mid !\overline{a}b)$ (processes $!\overline{a}b$ and $!\overline{a}c$ can either share a common resource $!a(x).\overline{x}$ - that sends a signal on the name it receives on a -, or have their own copy of this resource; note the shape of the normal form for the right hand side process):

```
> Check
Yes, size of the relation is 1
((^a)(!a(x).x[] | !a[c] | !a[b]),
(^e')(^a)(!a(x).x[] | !e'(x).x[] | !a[c] | !e'[b]));
```

Conclusion

We have developed some methods to automatically check bisimilarities between π-calculus processes, and shown their expressive power on a prototype implementation[2]. Our system is rather elementary, and cannot be compared as it is with other similar tools (like the Mobility Workbench [VM94], which is probably the closest to ours, Cesar/Aldebaran [FGK+96], the Jack Toolkit [ASS94], or the FC2tools package [AAVS96]). However, the possibility to reason on infinite states processes is specific to our tool[3], and represents an important feature in terms of expressiveness.

Our system can be made more robust by enriching the syntax (e.g. adding definitions of agents and the choice operator), and improving the bisimulation checking method (this means in particular modifying our algorithm to adopt a breadth-first strategy, instead of making "blind" recursive calls to the general bisimulation checking function. This would insure some kind of "computational completeness": if a finite up-to bisimulation relation exists, we find it after a finite number of steps).

Completeness of our methods is a key theoretical issue, as it is directly related to the understanding of the expressiveness of our system. One is interested in defining a class of terms (containing some infinite states processes) for which our algorithm is a decision procedure for bisimilarity. Let us explain informally where the difficulties come from: the key point is the up to parallel composition technique, that gives the possibility to reason with replicated terms. This technique is used to cancel common parallel components in two processes *right after* a transition has taken place. For example, take a process of the form $!a(\vec{b}).P$, liable to perform the transition $!a(\vec{b}).P \xrightarrow{a(\vec{c})} P' = !a(\vec{b}).P \mid P_{\vec{b}:=\vec{c}}$, and suppose $P \sim Q$, which implies $Q \xrightarrow{a(\vec{c})} Q'$: intuitively, the idea is that one should be able to cancel $P_{\vec{b}:=\vec{c}}$ *both* in P' and Q' (if we want to prove $P \sim Q$), otherwise P' and Q' could do the transitions $\xrightarrow{a(\vec{c'})}$, $\xrightarrow{a(\vec{c''})}$, ... , and the relation would keep growing *ad infinitum*. This phenomenon actually restricts considerably the freedom in the definition of the terms we can manipulate (hence the simplicity of the examples of Section 3), and it seems indeed that the up to parallel composition proof technique cannot be used as a brute force tool to prove bisimilarity results between infinite states processes. On the contrary, the idea is rather to use the automation of the up to techniques to *verify* a proof on paper, once we know that the up to techniques apply. Following this approach, work is in progress to adapt our methods to *open terms* [Ren97, Sim85], in order to be able to prove not only bisimilarity results, but also general bisimilarity laws (like for example the so-called *replication theorems*).

Another interesting direction could be the mechanisation of the proofs of this paper, reusing the work of [Hir97], which could allow one to extract a *certified* bisimilarity checker. Some more work has to be done in order to make these proofs tractable for the purpose of

[2] A beta version of the tool is available at `http://cermics.enpc.fr/~dh/pi/`

[3] We are aware of some efforts regarding the implementation of the up to techniques within the CONCUR project, but do not have details about the focus of this study and its outcome.

a theorem prover formalisation.

Acknowledgements Many thanks go to Michele Boreale for constant help during this study, as well as to Davide Sangiorgi for introducing the theoretical basis of it, and for insightful discussions.

References

[AAVS96] A.Bouali, A.Ressouche, V.Roy, and R.de Simone. The FC2 Toolset. demo presentation at TACAS'96, AMAST'96 and CAV'96, 1996.

[ASS94] A.Bouali, S.Larosa, and S.Gnesi. The integration project in the JACK Environement. *EATCS Bulletin*, (54), 1994.

[EG98] J. Engelfriet and T. Gelsema. Structural Inclusion in the pi-Calculus with Replication. Report 98-06, Leiden University, 1998.

[FGK+96] J.-C. Fernandez, H. Garavel, A. Kerbrat, R. Mateescu, L. Mounier, and M. Sighireanu. Cadp (caesar/aldebaran development package): A protocol validation and verification toolbox. In *Proceedings of CAV' 96*, volume 1102 of *LNCS - Springer Verlag*, pages 437–440, 1996.

[Hir97] D. Hirschkoff. A full formalisation of π-calculus theory in the Calculus of Constructions. In *Proceedings of TPHOL'97*, volume 1275, pages 153–169. LNCS, Springer Verlag, 1997.

[Hir98] D. Hirschkoff. Automatically Proving Up-to Bisimulation. Technical Report 98-123, CERMICS, Champs sur Marne, March 1998.

[Mil91] R. Milner. The polyadic π-calculus: a tutorial. Technical Report ECS-LFCS-91-180, October 1991. Also in *Logic and Algebra of Specification*, ed. F. L. Bauer, W. Brauer and H. Schwichtenberg, Springer-Verlag, 1993.

[PS96] M. Pistore and D. Sangiorgi. A partition refinement algorithm for the π-calculus. In Rajeev Alur, editor, *Proceedings of CAV '96*, volume 1102 of *LNCS*, 1996.

[PT87] R. Paige and R. E. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.

[Ren97] A. Rensink. Bisimilarity of open terms. In C. Palamidessi and J. Parrow, editors, *Expressiveness in Concurrency*, 1997. also available as technical report 5/97, University of Hildesheim, may 1997.

[San95] D. Sangiorgi. On the bisimulation proof method. Revised version of Technical Report ECS–LFCS–94–299, University of Edinburgh, 1994. An extended abstract can be found in Proc. of MFCS'95, LNCS 969, 1995.

[Sim85] R. De Simone. Higher-level synchronising devices in Meije-SCCS. *Theoretical Computer Science*, (37):245–267, 1985.

[SM92] D. Sangiorgi and R. Milner. Techniques of "weak bisimulation up to". In *CONCUR '92*, number 630 in LNCS, 1992.

[VM94] B. Victor and F. Moller. The Mobility Workbench — a tool for the π-calculus. In D. Dill, editor, *Proceedings of CAV'94*, volume 818 of *LNCS*, pages 428–440. Springer-Verlag, 1994.

State Spaces of Object-Oriented Petri Nets

Vladimír Janoušek Tomáš Vojnar

Department of Computer Science and Engineering, Technical University of Brno
Božetěchova 2, CZ–612 66 Brno, Czech Republic
e-mail: {janousek,vojnar}@dcse.fee.vutbr.cz

Abstract

This paper looks at object-oriented Petri nets, and considers the problem of analysing the state space, which in general distinguishes according to instance names of objects. It is proposed that most analysis can be done using a name-abstracted state space.

1 Introduction

Petri nets are a successful formalism which allows modelled systems to be described in a graphical way and, what is more, they offer a possibility of formal analysis of models based on them. However, basic Petri nets suffer from the lack of structuring. On the other hand, object-orientation provides a very powerful way of structuring programs and models. So, it should be advantageous to join these two approaches and obtain a modelling paradigm which would combine their advantages. In fact, several attempts to combine objects and Petri nets have already been done—see e.g. [LK94, SB94, Val96, ČJ97].

In this article, we will describe some results of our research on state space analysis methods for models based on object-oriented Petri nets (OOPNs). By OOPNs, we will understand the formalism developed at the computer science department of TU Brno and connected to the tool PNtalk [ČJV97]. This formalism is quite similar to the one of C. Sibetin-Blanc [SB94], from which our OOPNs differ mainly in describing classes by more than one net, since every OOPN class has one object net and several method nets.

In the following, we will focus especially on the problem of identifying objects and running methods and its impact on the notion of OOPNs' state spaces. We suppose the information about particular values of identifiers of objects and running methods to be an implementation detail, which should be abstracted away when defining states of running OOPNs. Therefore we will define the notion of name-abstracted state spaces, which can be subsequently used as a basis for adapting the concepts of occurrence graph analysis, as introduced by K. Jensen in the context of CPNs [Jen94], for the domain of OOPNs.

Problems with performing formal analysis connected to identifying dynamically appearing and disappearing instances of nets arise, however, not only in the area of the

OOPNs considered in this article, but they concern other object-oriented Petri net-based formalisms as well. Furthermore, we cannot get rid of these problems simply by using an algorithm for transforming object-oriented nets into plain high-level nets, as it was suggested e.g. in [SB94, LK94]. This is because in resulting nets there must appear a place or a construction generating identifiers which then become a distinguishing part of tuples representing tokens of originally different net instances folded together. Thus the problem of naming is carried into the domain of non-object nets and must be solved within their analysis process.

The remainder of this article begins with a short introduction to OOPNs, followed with a concise review of the key concepts of their formal definition. Then we discuss the above sketched questions connected to the notion of OOPNs' state spaces. We conclude the article with a suggestion of adopting Jensen's occurrence graph analysis methods for the area of OOPNs and we also briefly mention some other topics to be considered in the area of OOPNs' analysis in future.

2 The Basic Principles of OOPNs

The OOPN formalism[1] is characterized by a Smalltalk-based object-orientation enriched with concurrency and polymorphic transition execution, which allow for message sending, waiting for and accepting responses, creating new objects, and performing primitive computations.

OOPNs are based on viewing objects as active servers which offer reentrant services to other objects. Services provided by objects, as well as independent activities of objects, are described by Petri nets—services by *method nets*, object activities by *object nets*. Tokens in nets represent references to objects.

An OOPN consists of Petri nets organized in classes. Every class consists of an object net describing the internal activity of objects of this class and a set of dynamically instantiable method nets describing how these objects respond to messages. All method nets of a given class share access to the appropriate object net (places of the object net are accessible for transitions of method nets). Each method net has parameter places and a return place. Class inheritance is defined by the inheritance of object nets[2], together with sets of method nets[3]. Classes can also contain predicate methods which allow for atomic testing their objects' states without any side-effects.

Every object is either trivial (e.g. a number or a string) or it is an instance of some Petri net-described class consisting of one instance of the appropriate object net and several currently running instances of method nets. When an object receives a message, a new instance of the corresponding method net is created, parameters are put into the parameter places and the instance of the method net is being executed concurrently with all other

[1] A more detailed informal description of OOPNs can be found in [ČJV97].

[2] Inherited transitions and places identified by their names can be redefined and new places or transitions can be added.

[3] The implementation of inherited methods can be changed and new methods can be added.

net instances until the return place receives a token. Then the value of the token in the return place is passed to the message sender as the result of the requested service, and the instance of the method net is deleted. Message sending and object creations are specified as actions attached to transitions. Execution of transitions is polymorphic—invoked methods depend on classes of message receivers which are unknown at the compile time.

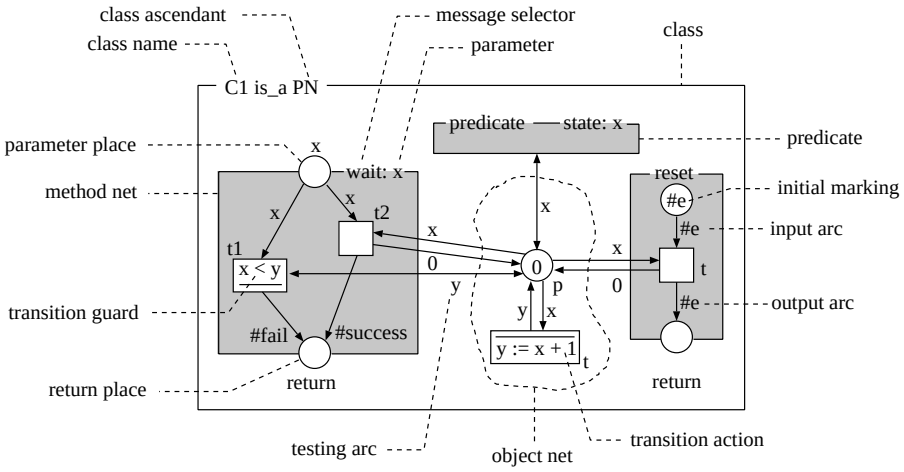


Figure 1: A simple class demonstrating the notion of the OOPN formalism

3 The Key Concepts of the OOPN Formal Definition

The entire definition of OOPNs can be found in [Jan97]. Here, we will only rephrase its basic ideas without making any effort to make this description formal and complete.

The definition distinguishes primitive and non-primitive objects. Non-primitive objects are specified by their Petri net-described classes and have states which can be potentially changed during the system evolution either by the objects' own internal activity or by executing their methods. On the other hand, primitive objects are constants, such as numbers, booleans, symbols, etc., which are simply available via their names. Their methods are atomically evaluable and do not have any side-effects.

Let us suppose the existence of a universe $U = CONST \cup NAME \cup CLASS$, where $CONST$ is a set of primitive objects, $NAME$ is a set of names intended for identifying non-primitive objects which can be created and discarded dynamically at the runtime, and $CLASS$ is a finite set of class names. The sets $CONST$, $NAME$, and $CLASS$ are pairwise disjoint. Elements of the universe are called atoms. Multisets of n-tuples of atoms play the role of markings of places. Each class name $c \in CLASS$ has the domain $Dom(c) \subseteq NAME$ which represents the set of all potential instances of this class. Domains are pairwise disjoint. Let us further suppose the existence of a set of message selectors MSG and a set of special message selectors MSG_S, $MSG_S \cap MSG = \emptyset$. Special message

selectors are reserved for special methods, such as the object identity operator ==, which are atomically evaluable and do not depend on objects' states—they work over objects' names only.

Next, let us have a look at the inscription language. Expressions of this language are either terms (i.e. constants, names of classes, or variables from some set VAR), message sending constructions $e_0.msg(e_1, e_2, ..., e_m)$, where e_i is a term and $msg^{(m)} \in (MSG \cup MSG_S)$ is a message selector with the arity $m \geq 0$, or formal sums of terms. Formal sums of terms form arc expressions and evaluate to multisets of n-tuples. Message sending can be used in transition guards and actions.

Depending on the message selector and on the class of the receiver, a message sending expression can be interpreted in two ways. If the receiver is a primitive object, or if the message selector is a special message selector, we speak about a primitive message sending evaluation, which is interpreted as a function and has no side effects, i.e. it does not change the state of any object and it works over constants and names only. Otherwise, we encounter a non-primitive message sending evaluation which will change the state of the running OOPN by launching a new method net instance.

The structure of OOPNs is defined as follows. An OOPN is a system of classes with a particular initial class and some initial object with a predefined name. A system of classes comprises a system of names and primitive objects and, for every class, a specification of the structure of its instances. Over a system of classes, the inheritance relation can be defined. A specification of the structure of instances of a given class consists of an object net, a set of method nets, a set of predicates, a set of selectors corresponding to the method nets and predicates, and, finally, a specification of possible names of instances of the object net and method nets. Every class must imply the so-called system of nets, in which method nets share places with the object net. A system of nets is defined as a set of nets whose potential instances are identified in such a way that it is always apparent to which net they belong. Object nets consist of places and transitions. Every place has some initial marking. Every transition has conditions (i.e. inscribed testing arcs), preconditions (i.e. inscribed input arcs), a guard, an action, and postconditions (i.e. inscribed output arcs). Method nets are similar to objects nets but, in addition, every one of them has a set of parameter places and an output place. Every predicate embodies a set of conditions over places of the appropriate object net, a guard, and a set of parameters.

The specification of the dynamic behaviour of OOPNs is based on the definition of net instances. Every net instance entails its identification (i.e. name) and a marking of its places and transitions. A marking of a place is a multiset of n-tuples over constants, class names, and object identifiers. A transition marking is a multiset of invocations. Every invocation contains an identifier of the invoked net instance and a binding of the input variables of the appropriate transition. An object is a system of net instances which contains exactly one instance of the appropriate object net and a set of currently running instances of method nets. A state of a run of an OOPN-based model corresponds to a system of objects, i.e. to a set of up-to-date objects. States can be changed by events corresponding to transition occurrences. There are four kinds of them: A (atomic action), N (new object instantiation), F (fork, i.e. new method net instance invocation), and J

(join, i.e. termination of the run of a method net instance). Every event is a 4-tuple (e, id, t, b) including a specification e of its type (A, N, F, or J), the identifier id of the net instance in which it occurred, the name of the transition t it is bound to, and a binding b of the variables of this transition. If an event (e, id, t, b) occurs in a state S and changes it into a state S', we speak about a step $S[e, id, t, b\rangle S'$. The set of all systems of objects is denoted as SO and the set of all events as EV.

4 The Notion of the State Space of OOPNs

In this section, we will introduce the notion of OOPNs' state spaces and we will very briefly sketch how it can be used for adapting the Jensen's occurrence graph theory to make it work with OOPNs.

We will start our discussion of the state space of OOPNs by presenting a modified version of the definition of the (total) state space from [Jan97].

Definition 4.1

1. *The* ***set of systems of objects reachable from a given system of objects*** S *(denoted as* $[S\rangle$ *in the following) is the smallest set of systems of objects, such that:*
   (a) $S \in [S\rangle$.
   (b) *If* $S' \in [S\rangle$ *and* $S'[e, id, t, b\rangle S''$ *for some* $(e, id, t, b) \in EV$, *then also* $S'' \in [S\rangle$.
2. *The* ***(total) state space*** $[S_0\rangle$ *of an object-oriented Petri net OOPN with an initial system of objects* S_0 *is the set of systems of objects reachable from the initial system of objects* S_0.
3. *The notion of the set of systems of objects reachable from a given starting system of objects can be easily extended to work with sets of starting systems of objects as well. Suppose that we are given such a set of systems of objects* $\mathbb{S}$, *then we can define* $[\mathbb{S}\rangle = \bigcup_{S \in \mathbb{S}} [S\rangle$.

The total state space from the above definition differentiates even among systems of objects differing only in the names of the involved net instances. However, is this really necessary? It seems that not and that it is always possible to neglect concrete identifications of net instances by using a suitable renaming equivalence relation over systems of objects. We can advocate this idea in the following way. Firstly, concrete names of net instances cannot influence the future evolution of systems of objects in any way (apart from renaming) because there is no possibility of accessing them. Furthermore, it also does not appear to be sensible to differentiate systems of objects equal up to renaming only because their histories cannot be made equal even when applying renaming. If we are interested in one particular history of some system of objects, we can always subsequently concentrate on it and ignore all other possible histories of the same system of objects (up to renaming).

So using names of net instances appears to be an implementation detail which is necessary to allow the manipulation with net instances and their relations, but the concrete values of these names are not interesting from the point of view of analysing properties of modelled systems. Therefore, in the following paragraphs, we will semi-formally describe the renaming equivalence relation and exploit it for a subsequent definition of the name-abstracted state space of an OOPN. The notion of the name-abstracted state space will then serve as a basis for further considerations about evaluating properties of OOPN-based models.

Definition 4.2 *Systems of objects $S_1, S_2 \in SO$ are **equivalent up to renaming** ($S_1 \sim S_2$), iff there exists a bijection Π on the universe U, such that the following is true:*

1. *$\forall x \in CONST \cup CLASS : x = \Pi(x)$, i.e. constants and names of classes are mapped onto themselves.*
2. *For all net instance identifiers id involved in S_1 (i.e. present in the marking of places and transitions of S_1):*
   (a) *The nets corresponding to id in S_1 and to its renaming $\Pi(id)$ in S_2 are the same.*
   (b) *The marking of the corresponding places and transitions of the net instances identified by id in S_1 and $\Pi(id)$ in S_2 is the same up to the application of the renaming Π to all non-trivial objects in the marking of the net instance corresponding to id in S_1.*

Remark 4.1 *We will use the notation $\langle S \rangle$ to denote the equivalence class of elements from SO which contains the element S. Normally, the notation $[S]$ would be used but we will reserve it for "high-level" equivalence classes over the name-abstracted state space. Such equivalence classes can be defined by modellers or derived from specifications of symmetries. We will further refer to the name equivalence classes of $\sim$ over SO also as to the name-abstracted systems of objects and we will be printing them by means of the "black board alphabet", i.e. $\mathbb{S}$, $\mathbb{S}_1$, etc. Finally, the quotient of SO by $\sim$ will be written as $SO_\sim$, i.e. $SO_\sim = SO \backslash \sim$.*

Now we are ready to define the notion of the name-abstracted state space. Its elements will be renaming equivalence classes of the set of states reachable from every possible renaming of a given initial system of objects. So the name-abstracted state space can be defined as the quotient of the corresponding total state space by $\sim$ extended by taking into account every renaming of the initial system of objects.

Definition 4.3

1. *The **name-abstracted set of systems of objects reachable from the given name-abstracted system of objects** $\langle S \rangle$ (denoted as $[\langle S \rangle\rangle_\sim$) is defined as the quotient of $[\langle S \rangle\rangle$ by the renaming equivalence relation $\sim$, i.e. $[\langle S \rangle\rangle_\sim = [\langle S \rangle\rangle \backslash \sim$.*

2. *The* ***name-abstracted state space*** $[\langle S_0 \rangle\rangle_\sim$ *of an object-oriented Petri net with the total state space* $[S_0\rangle$ *is defined as the name-abstracted set of systems of objects reachable from the given initial name-abstracted system of objects.*

Now it would be possible to show that the elements of a given name-abstracted state space are complete equivalence classes of systems of objects from SO with respect to the renaming equivalence relation, but we will omit the proof of this property.

Before proceeding further on we have to extend the notion of the renaming equivalence in order to make it work also with events, i.e. transition occurrences. This step is necessary for the definition of O-graphs of OOPNs which will be presented later.

Definition 4.4 *Events* $E_1 = (e_1, id_1, t_1, b_1)$ *and* $E_2 = (e_2, id_2, t_2, b_2)$ *from* EV *are* ***equivalent up to renaming*** *(*$E_1 \sim E_2$*), iff there exists a bijection* Π *on the universe* U*, such that:*

1. $\forall x \in CONST \cup CLASS : x = \Pi(x)$*, i.e. constants and names of classes are mapped onto themselves.*
2. *For all net instance identifiers id involved in the event* E_1 *(i.e.* id_1 *plus identifiers of the non-trivial objects used in the binding* b_1*), the nets corresponding to id in* E_1 *and to its renaming* $\Pi(id)$ *in* E_2 *are the same.*
3. *The types of* E_1 *and* E_2 *are the same, i.e.* $e_1 = e_2$*.*
4. $id_2 = \Pi(id_1)$*.*
5. *The involved transitions are the same, i.e.* $t_1 = t_2$*.*
6. *The binding of variables* b_1 *in* E_1 *is the same as the one of* E_2 *up to the* Π*-renaming of the non-trivial objects involved in* b_1*.*

Remark 4.2 *We will use the notation* $\langle E \rangle$ *to denote the equivalence class of elements from* EV *which contains the element* E*. Moreover, we will further refer to the name equivalence classes of* $\sim$ *over* EV *also as to the name-abstracted events and we will be printing them by means of the "black board alphabet", i.e.* $\mathbb{E}$*,* $\mathbb{E}_1$*, etc. Finally, the quotient of the set* EV *by* $\sim$ *will be denoted as* $EV_\sim$ *in the following.*

We should note that two events equivalent up to renaming can lead from the same starting system of objects into two different final systems of objects even when taking into account renaming. This is because in events there is information only about classes of the involved objects, but not about their states. We can easily find a system of objects in which the same place contains two objects of the same class, but in different states. Let these objects be movable to another place by the same transition. Then, by using two events equal up to renaming, we can immediately obtain two different final systems of objects. However, this property of name-abstracted events should not cause any problems in the forthcoming considerations.

Finally, we will define the notion of name-abstracted steps, finite occurrence sequences, and infinite occurrence sequences. These three terms formalize relations between name-abstracted systems of objects and name-abstracted events.

Definition 4.5

1. *A **name-abstracted step** is a triple* $\langle S_1\rangle[\langle e, id, t, b\rangle\rangle\langle S_2\rangle$, *such that* $\langle S_1\rangle, \langle S_2\rangle \in SO_\sim$, $\langle e, id, t, b\rangle \in EV_\sim$, *and there exist* $S \in \langle S_1\rangle$, $S' \in \langle S_2\rangle$, *and* $(e, id', t, b') \in \langle e, id, t, b\rangle$, *such that* $S[e, id', t, b'\rangle S'$.
2. *A **finite name-abstracted occurrence sequence of the length** n is a finite sequence of name-abstracted steps* $\mathbb{S}_1[\mathbb{E}_1\rangle\mathbb{S}_2[\mathbb{E}_2\rangle \ldots \mathbb{S}_n[\mathbb{E}_n\rangle\mathbb{S}_{n+1}$.
3. *An **infinite name-abstracted occurrence sequence** is an infinite sequence of name-abstracted steps* $\mathbb{S}_1[\mathbb{E}_1\rangle\mathbb{S}_2[\mathbb{E}_2\rangle \ldots$.

Remark 4.3 *Note that we do not require name-abstracted steps to fulfil the following condition:* $\forall S \in \langle S_1\rangle, S' \in \langle S_2\rangle, (e, id', t, b') \in \langle e, id, t, b\rangle : S[e, id', t, b'\rangle S'$.

In the end of this section, we will show that the above defined name-abstracted state spaces allow us to very simply adapt the theory of occurrence graphs proposed by K. Jensen in [Jen94] for the domain of much more dynamic OOPNs. In fact, all the definitions and propositions we have to formulate are just simple extensions of the original ones of K. Jensen. We will demonstrate this on the definition of OOPNs' full occurrence graphs, which provide a basic representation of OOPNs' name-abstracted state spaces.

Definition 4.6 (cf. Def. 1.3 of [Jen94]) *The **full occurrence graph** of an OOPN, which we will also denote as the **O-graph**, is defined as the directed graph* $OG = (V, A, N)$ *with vertices* V, *arcs* A, *and the node function* N, *such that:*

1. $V = [\langle S_0\rangle\rangle_\sim$.
2. $A = \{(\mathbb{S}_1, \mathbb{E}, \mathbb{S}_2) \in V \times EV_\sim \times V \mid \mathbb{S}_1[\mathbb{E}\rangle\mathbb{S}_2\}$.
3. $\forall a = (\mathbb{S}_1, \mathbb{E}, \mathbb{S}_2) \in A : N(a) = (\mathbb{S}_1, \mathbb{S}_2)$.

Because of space limitations, we will not continue discussing OOPNs' O-graphs, but, from their definition, it should be obvious that the subsequent definitions and propositions working over them, such as proof rules or the notions of equivalences and symmetries, could also be obtained as simple extensions of the ones defined in the context of the Jensen's CPNs.

5 Conclusions and Future Research

In the article, we discussed problems accompanying the notion of state spaces of object-oriented Petri nets connected to the tool PNtalk, which, however, seem to arise also in other Petri net-based formalisms exploiting dynamic instantiation of nets.

We have defined the notion of total state spaces of OOPNs which are so detailed that even concrete names of net instances are important. Then we have argued that it is probably not necessary to take into account particular names of net instances within the appropriate systems of objects. Therefore we have suggested the notion of name-abstracted

state spaces. Name-abstracted state spaces have been proposed in such a way that all concepts developed for the state space analysis of CPNs in [Jen94] should be applicable also in the area of OOPNs. This has been demonstrated on the definition of O-graphs. However, we hope that we will be able to adopt other concepts, such as SCC-graphs, OE-graphs, OS-graphs, and the corresponding proof rules, in a similarly straightforward way.

Nevertheless, there remain some problems concerning the state space analysis of OOPNs. It is especially the problem of the complexity of testing systems of objects to be equivalent up to renaming when building O-graphs. For the sake of implementing such an algorithm, every name-abstracted system of objects can be represented by some concrete system of objects which is a member of the appropriate equivalence class. Unfortunately, systems of objects have the form of labeled oriented graphs and the test of their equivalence up to renaming leads to their unification, which is, in general, exponentially hard in the number of nodes and arcs. So we have to find some additional property of systems of objects which would allow us to decrease the complexity of the tests of their equivalence up to renaming. If it turns out that it is impossible to decrease the efficiency of the renaming equivalence tests in the context of general OOPNs, we should try to propose a suitable subclass of OOPNs which would allow it. If we e.g. restrict ourselves to using 1-safe OOPNs (i.e. at most one object in every place instance and at most one invocation of every transition instance at a time), we will probably be able to unify systems of objects with respect to some suitable order over places and transitions and the renaming equivalence test will become almost trivial.

Over the proposed name-abstracted state spaces, we would further like to propose a suitable temporal logic for describing properties of OOPN-based models. This logic could be similar to some of the logics proposed for non-object Petri nets, such as ASK-CTL for CPNs [CCM96], but it should also allow modellers to express properties specific for the object-oriented world, such as relations among objects, message sending, waiting for and accepting responses, etc. For the proposed logic, we will have to develop a suitable model checking procedure fighting the state space explosion in some way. Here, we can consider using SCC-graphs or stubborn sets. Further, we should try to exploit the modularity of OOPN-based models in some way, for example by using some analogy to the modular O-graphs [CP95] or facilitating some kind of compositional reasoning, as e.g. the one in [Kin97]. Here there will most likely appear new problems to be solved connected to changing relations among net instances.

It is also necessary to create some computer support tool for the state space analysis of OOPNs. But, it could be also interesting to try to use some existing tool for high-level nets. Here, some transforming procedure would be necessary and it would be also vital to implement some name-abstracting mechanism working over the resulting nets.

Acknowledgment. This work was supported by grants of the Czech Grant Agency under the contracts 102/98/0552 "Research and Applications of Heterogeneous Models" and 102/96/0986 "Object-Oriented Database Model".

References

[CCM96] A. Cheng, S. Christensen, and K.H. Mortensen. Model Checking Coloured Petri Nets Exploiting Strongly Connected Components. In M.P. Spathopoulos, R. Smedinga, and P. Kozák, editors, *Proceedings of the International Workshop on Discrete Event Systems (WODES'96)*, pages 169–177, Edinburgh, Scotland, UK, August 1996.

[ČJ97] M. Češka and V. Janoušek. A Formal Model for Object-Oriented Petri Nets Modeling. *Advances in Systems Science and Applications, An Official Journal of the International Institute for General Systems Studies*, Special Issue:119–124, 1997.

[ČJV97] M. Češka, V. Janoušek, and T. Vojnar. PNtalk – A Computerized Tool for Object-Oriented Petri Nets Modelling. In F. Pichler and R. Moreno-Díaz, editors, *Proceedings of the 5th International Conference on Computer Aided Systems Theory and Technology – EUROCAST'97*, volume 1333 of *Lecture Notes in Computer Science*, pages 591–610, Las Palmas de Gran Canaria, Spain, February 1997. Springer-Verlag.

[CP95] S. Christensen and L. Petrucci. Modular State Space Analysis of Coloured Petri Nets. In G. De Michelis and M. Diaz, editors, *Proceedings of the 16th International Conference on Application and Theory of Petri Nets*, volume 935 of *Lecture Notes in Computer Science*, pages 201–217, Turin, Italy, June 1995. Springer-Verlag.

[Jan97] V. Janoušek. Objektově orientované Petriho sítě – formální základ jazyka PNtalk. Technical report, Department of Computer Science and Engineering, Technical University of Brno, Czech Republic, January 1997. (In Czech).

[Jen94] K. Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use, Vol. 2: Analysis Methods.* EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1994.

[Kin97] E. Kindler. A Compositional Partial Order Semantics for Petri Net Components. In P. Azéma and G. Balbo, editors, *Proceedings of the 18th International Conference on Application and Theory of Petri Nets*, volume 1248 of *Lecture Notes in Computer Science*, pages 235–252, Toulouse, France, June 1997. Springer-Verlag.

[LK94] C.A. Lakos and C.D. Keen. LOOPN++: A New Language for Object-Oriented Petri Nets. Technical Report R94-4, Department of Computer Science, University of Tasmania, Hobart, Tasmania 7001, April 1994.

[SB94] C. Sibertin-Blanc. Cooperative Nets. In R. Valette, editor, *Proceedings of the 15th International Conference on Application and Theory of Petri Nets*, volume 815 of *Lecture Notes in Computer Science*, pages 471–490, Zaragoza, Spain, June 1994. Springer-Verlag.

[Val96] R. Valk. On Processes of Object Petri Nets. Technical Report FBI-HH-B-185/96, Fachbereich Informatik, Universität Hamburg, 1996.

The essence of Petri nets and transition systems through Abelian groups

Gabriel Juhás

*Institute of Control Theory and Robotics, Slovak Academy of Sciences
Dúbravská 9, 842 37 Bratislava, Slovakia
email: utrrjuhi@savba.sk, gabrielj@brics.dk

Abstract

In the paper we investigate the role of Abelian groups in description of Petri nets and transition systems. We study the extension of Petri nets based on generalising algebra used in the dynamics of nets. We show that (partial) algebras embeddable to Abelian groups play an important role in preserving some natural properties such as determinism and commutativity of transition occurrences, state independence of the change caused by transition occurrences, and advantages given by the state equation.

1 Introduction

Petri nets are one of the first well established and widely used non-interleaving models of concurrent systems. Their origin arisies from Carl Adam Petri's dissertation [Pet62] and the later concept of vector addition systems [KM69]. Petri nets are popular and successfully used in many practical areas.

A Petri net is given by a set of places representing system components, a set of transitions representing a set of atomic actions of the system, and their relationship given by an input and output function associating with each transition a multi-set over a set of places [Pet73, Pet81, MM90]. A state of a net is a multi-set over the set of places called a marking. An occurrence of a transition removes/adds tokens from/to the marking according to the input/output function, respectively. A transition is enabled to occur iff in every place there are enough tokens to fire. Labelled transition systems [Kel76] represent the basic interleaving model of concurrency [WN95]. They may be described as a directed graph whose nodes are system

*The part of this work was done during the author's visit at BRICS (Basic Research in Computer Science, Centre of the Danish National Research Foundation) Department of Computer Science, University of Aarhus, Ny Munkegade, DK-8000 Aarhus C, Denmark.

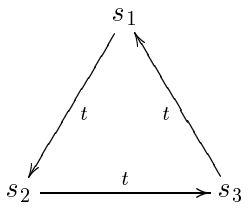
states and arcs, called transitions, are labelled by elements from a set of labels representing a set of atomic actions of the system. For a Petri net, the related transition system, called sequential case graph, is the graph whose nodes are markings of the Petri net and whose arcs are labelled by transitions of the Petri net. Thus, one should remember that the term 'transition' is used in Petri nets in the different meaning from transition systems.

A lot of effort has been put into the abstraction (see *e.g.* [Win87, MM90]) and extension of Petri nets. One can extend Petri nets by modifying the transition enabling rule in order to have a more expressive model, for example we can mention nets with inhibitor arcs that allow testing of zero markings [Pet81, JK91]. Others can introduce types of tokens, such nets are commonly called high level nets [JR91]. From a purely modelling point of view, high level nets may be viewed only as a more comprehensive model, that can be described by several low level nets.

The main advantages of Petri nets consist of the fact that they grant a well arranged graphical expression of the system as well as a simple model in the form of a linear algebraic system. Petri nets have several important properties:

- the occurrences of transitions are *deterministic*;
- the occurrences of transitions are *commutative* in the meaning that if two sequences of transitions with the same number of occurrences of single transitions are enabled in a marking then their occurrences lead to the common new marking;
- and, finally, the occurrences of sequences of transitions are *consensual* in the meaning that if the occurrences of two sequences of transitions in one state (marking) lead to the common new state, then they lead to the common new state from every other state in which both sequences are enabled to occur.

However, there are some behavioural limitations of Petri nets. Take, for example, the labelled transition system $\mathcal{S}$ with three states $\{s_1, s_2, s_3\}$ and just one label, *i.e* just one atomic action t, given as follows:



Clearly, there does not exist any Petri net whose sequential case graph is (isomorphic to) the given transition system $\mathcal{S}$, because for an arbitrary set of places P all nonzero elements of the free commutative monoid over P have infinite order. In other words, the system S cannot be (directly, *i.e.* using just one transition) modelled by any Petri net.

This leads us to an interesting question: Can Petri nets be extended to model wider class of systems including such systems as $\mathcal{S}$, while preserving the advantages and properties mentioned above, i.e. determinism, commutativity and consensuality of occurrences of transitions? And if yes, which class is the biggest class of systems that can be described as those extended Petri nets? The answer is related to the algebra used in the Petri net dynamics.

Starting with the very general setting, in this paper we study the possibility of extension of Petri nets with the aim of preserving properties of commutativity and consensuality of occurrences of transitions. After we show that existing extensions generally do not preserve these properties, we show that using of a (partial) algebra that can be embedded to an Abelian group is the limitation of those efforts.

2 Petri nets - the state of the art

In the first subsection of this section we review basic formal definitions of place/transition nets (p/t nets) and labelled transition systems. Then we deal with some extensions of p/t nets found in the literature.

2.1 Basic definitions

Before the defining of Petri nets, let us first define some notation.

We use $\mathbb{Z}$ to denote integers, $\mathbb{Z}^+$ to denote positive integers, and $\mathbb{N}$ to denote nonnegative integers.

Moreover, we also shortly write $\mathbb{Z}$ to denote the infinite cyclic group of integers with addition $(\mathbb{Z}, +)$, and $\mathbb{N}$ to denote the commutative monoid of nonnegative integers with addition $(\mathbb{N}, +)$.

Given two arbitrary sets, say A and B, symbol B^A denotes the set of all functions from A to B. Given a function f from A to B and a subset C of the set A we write $f|_C$ to denote the restriction of the function f on the set C.

To denote the free commutative monoid over a set A, *i.e.* set of all multi-sets over A with multi-set addition we write $(\mathbb{N}^A, +)$ or shortly $\mathbb{N}^A$. Let $\mathbb{N}^A_{fin} = \{b \mid b \in \mathbb{N}^A \wedge |A_b| \in \mathbb{N}\}$, where $A_b = \{a \mid a \in A \wedge b(a) \neq 0\}$ is a subset of the set A mapped by function $b \in \mathbb{N}^A$ to non-zero integers, *i.e.* A_b is the set of elements from A that are contained (occur at least once) in the multi-set b. So, $\mathbb{N}^A_{fin}$ is the set of all finite multi-sets over the set A. Clearly, $(\mathbb{N}^A_{fin}, +|_{\mathbb{N}^A_{fin} \times \mathbb{N}^A_{fin}})$ is a submonoid of the monoid $(\mathbb{N}^A, +)$. Similarly, to denote the free Abelian group over a set A we write $(\mathbb{Z}^A, +)$ or shortly $\mathbb{Z}^A$. Let $\mathbb{Z}^A_{fin} = \{b \mid b \in \mathbb{Z}^A \wedge |A_b| \in \mathbb{N}\}$, where A_b is given as previously. Evidently, $(\mathbb{Z}^A_{fin}, +|_{\mathbb{Z}^A_{fin} \times \mathbb{Z}^A_{fin}})$ is a subgroup of the free Abelian group $(\mathbb{Z}^A, +)$. As usual, we write only $(\mathbb{N}^A_{fin}, +)$ or shortly $\mathbb{N}^A_{fin}$ instead of rather complicated $(\mathbb{N}^A_{fin}, +|_{\mathbb{N}^A_{fin} \times \mathbb{N}^A_{fin}})$; and $(\mathbb{Z}^A_{fin}, +)$ or shortly $\mathbb{Z}^A_{fin}$ instead of $(\mathbb{Z}^A_{fin}, +|_{\mathbb{Z}^A_{fin} \times \mathbb{Z}^A_{fin}})$. Notice that

if the set A is finite, then $\mathbb{N}^A = \mathbb{N}^A_{fin}$ and $\mathbb{Z}^A = \mathbb{Z}^A_{fin}$.
As one may see from the previous notation, we often use the symbol + in the paper universally to denote a binary operation, *i.e.* we use the symbol + for different operations.

According to the previous section, in algebraic form Petri nets (place/transition nets or shortly p/t nets) [Pet73, Pet81, MM90] are defined as follows:

Definition 2.1.1 *A place/transition net is an ordered tuple $\mathcal{N} = (P, T, I, O)$, where P and T are non-empty distinct sets of places and transitions; $I : T \to \mathbb{N}^P$ is an input function; $O : T \to \mathbb{N}^P$ is an output function. A marked place/transition net is an ordered tuple $\mathcal{MN} = (\mathcal{N}, M_0)$, where $\mathcal{N}$ is a p/t net; and $M_0 \in \mathbb{N}^P$ is an* initial marking.

A state of a p/t net called marking and denoted by M is a multi-set over P, *i.e.* an element of $\mathbb{N}^P$. The dynamics of the net is expressed by the occurrence (firing) of enabled transitions. A transition $t \in T$ is enabled to occur in a marking $M \in \mathbb{N}^P$ iff $\forall p \in P : M(p) \geq I(t)(p)$, *i.e.* iff $\exists X \in \mathbb{N}^P : X + I(t) = M \in \mathbb{N}^P$. Occurrence of an enabled transition $t \in T$ in a marking M then leads to the new marking M' given by $M' = X + O(t)$ for the $X \in \mathbb{N}^P$ such that $X + I(t) = M$, *i.e.* to $M' = M + O(t) \Leftrightarrow I(t)$.
It is sometimes usual to restrict the multiplicity of places (number of tokens) in a p/t net by a capacity. So we have:

Definition 2.1.2 *A p/t net with capacity is an ordered tuple $\mathcal{KN} = (\mathcal{N}, K)$, where $\mathcal{N} = (P, T, I, O)$ is a p/t net and $K : P \to \mathbb{N} \cup \{\infty\}$ is a capacity function.*

The capacity function restricts the set of markings and the enabling rule as follows. Given a $\mathcal{KN} = (\mathcal{N}, K)$, a marking M of $\mathcal{N}$ is a marking of $\mathcal{KN}$ iff $\forall p \in P : M(p) \leq K(p)$, and therefore a transition $t \in T$ is enabled to occur in a marking M of $\mathcal{KN}$ iff $\forall p \in P : (M(p) \geq I(t)(p)) \ \wedge \ (M(p) + O(t)(p) \Leftrightarrow I(t)(p) \leq K(p))$.
Clearly, the class of all p/t nets is a subclass of the class of all p/t nets with capacity, where $K(p) = \infty$ for each $p \in P$ in a p/t net.

At this point we recall the definition of labelled transition systems [WN95]:

Definition 2.1.3 *A labelled transition system is an ordered tuple $\mathcal{S} = (S, L, \Leftrightarrow)$, where S is a set of states, L is a set of labels and $\Leftrightarrow \subseteq S \times L \times S$ is a transition relation.*

The fact that $(s, a, s') \in \Leftrightarrow$ is written as $s \overset{a}{\Leftrightarrow} s'$. Denoting by L^* the monoid of all finite strings of labels from L with concatenation, it is obvious to extend the transition relation to *string transition relation* $\Leftrightarrow_\star \subseteq S \times L^* \times S$ as follows: $(s, q, s') \in \Leftrightarrow_\star$ whenever there exists a, possibly empty, string of labels $q = a_1 ... a_n$ such that $s \overset{a_1}{\Leftrightarrow} s_1 \cdots \overset{a_n}{\Leftrightarrow} s'$. To denote $(s, q, s') \in \Leftrightarrow_\star$ we simply write $s \overset{q}{\Leftrightarrow}_\star s'$. Clearly, $s \overset{q}{\Leftrightarrow}_\star s' \overset{v}{\Leftrightarrow}_\star s'' \Longrightarrow s \overset{qv}{\Leftrightarrow}_\star s''$. As

usual, we also write $s \overset{a}{\Leftrightarrow\!\!\rightarrow}$ and $s \overset{q}{\Leftrightarrow\!\!\rightarrow}_\star$ to denote that there exist $s \in S$ such that $s \overset{a}{\Leftrightarrow\!\!\rightarrow} s'$ and $s \overset{q}{\Leftrightarrow\!\!\rightarrow}_\star s'$, respectively. A state s' is said to be reachable from a state s, iff there exists a string of labels q such that $s \overset{q}{\Leftrightarrow\!\!\rightarrow}_\star s'$. Given a state $s \in S$, the set of all states reachable from s is denoted by $\{s \Leftrightarrow\!\!\rightarrow_\star\}$. A labelled transition system is said to be reachable iff every $s' \in S$ is reachable from a fixed state $s \in S$ (*i.e.* $\exists s \in S : \{s \Leftrightarrow\!\!\rightarrow_\star\} = S$). A labelled transition system $\mathcal{S} = (S, L, \Leftrightarrow\!\!\rightarrow)$ is called deterministic iff $\forall s \overset{a}{\Leftrightarrow\!\!\rightarrow} s', s \overset{a}{\Leftrightarrow\!\!\rightarrow} s'' : s' = s''$.

Definition 2.1.4 *A pointed labelled transition system is an ordered tuple $\mathcal{PS} = (\mathcal{S}, i)$ where $\mathcal{S} = (S, L, \Leftrightarrow\!\!\rightarrow)$ is a labelled transition system; and $i \in S$ is a distinguished initial state such that $\{i \Leftrightarrow\!\!\rightarrow_\star\} = S$,* i.e. *every state is reachable from i.*

Now it is straightforward to see that each p/t net $\mathcal{N}$ can be associated with a deterministic transition system.

Definition 2.1.5 *Let $\mathcal{N} = (P, T, I, O)$ be a p/t net. Then the labelled transition system $\mathcal{S} = (\mathbb{N}^P, T, \Leftrightarrow\!\!\rightarrow)$ such that $M \overset{t}{\Leftrightarrow\!\!\rightarrow} M' \iff \Big(t$ is enabled to occur in M and $M' = M + O(t) \Leftrightarrow I(t)\Big)$ is called* sequential case graph *of the p/t net $\mathcal{N}$. Given any marking $M_0 \in \mathbb{N}^P$, the pointed labelled transition system $\mathcal{PS} = (\{M_0 \Leftrightarrow\!\!\rightarrow_\star\}, T, \Leftrightarrow\!\!\rightarrow \cap \{M_0 \Leftrightarrow\!\!\rightarrow_\star\} \times T \times \{s \Leftrightarrow\!\!\rightarrow_\star\}, M_0)$ is called* sequential case graph *of the marked p/t net $\mathcal{MN} = (\mathcal{N}, M_0)$.*

As follows, for a finite sequence $q = a_1 \dots a_n$ over a set A we write b_q to denote Parikh's image of q, *i.e.* $b_q \in \mathbb{N}^A$ is a multi-set in which the number of the occurrences $b_q(a)$ of each element a from A is given by the number of its occurrences in q, formally $b_q(a) = |\{i \mid i \in \{1, \dots, n\} \wedge a_i = a\}$ for every $a \in A$.

Moreover, given a function $f : T \to \mathbb{Z}^P$, we denote by $\hat{f}$ the linear $\mathbb{Z}$-extension of the function f, *i.e.* we have $\hat{f} : \mathbb{Z}^T_{fin} \to \mathbb{Z}^P$ is such that $\forall b \in \mathbb{Z}^T_{fin} : \hat{f}(b) = \sum_{t \in T_b} f(t) \cdot b(t)$, where naturally the sum of empty set is zero-function, *i.e.* we define $\hat{f}(0) = 0$. In other words, for finite sets P and T, $\hat{f}$ may be understood as a 'matrix' whose rows are values $f(t)$, and then, for given 'vector' b, value $\hat{f}(b)$ represents the result of standard multiplication of the 'matrix' $\hat{f}$ by 'vector' b.

Properties of the free Abelian group $\mathbb{Z}^P$ over the set P enable us to write $M' = M + \hat{C}(b_q)$ whenever $M \overset{q}{\Leftrightarrow\!\!\rightarrow}_\star M'$. Moreover, existence of a solution of the equation $\hat{C}(Y) = M' \Leftrightarrow M$ in $\mathbb{N}^T_{fin}$ is a necessary condition of reachability of M' from M. The solution $Y \in \mathbb{N}^T_{fin}$ then determines the number of transition occurrences that lead from M to M'. One usually calls the equation $M' = M + \hat{C}(Y)$ or $M' = M + \hat{O}(Y) \Leftrightarrow \hat{I}(Y)$ a *state equation* of p/t nets and $Y \in \mathbb{N}^T_{fin}$ a *firing vector*. Thus, in the p/t nets and their sequential case graphs the change of the state is invariant on the order of transition occurrences and depends only on the number of their occurrences.

In this place we give a definition of elementary nets, because they represent a natural example of nets in which a partial groupoid, namely the powerset 2^P of the set of places P with distinct union $\uplus$ as the partial operation, is used.

Definition 2.1.6 *An elementary net is an ordered tuple* $\mathcal{EN} = (P, T, I, O)$ *where* P *and* T *are disjoint sets of places and transitions and* $I, O : T \to 2^P$ *are input and output functions.*

A marking of an elementary net $\mathcal{EN}$ is a subset of P. A transition $t \in T$ is enabled to occur in a marking $M \in 2^P$ iff $\exists X \subseteq P : X \uplus I(t) = M \ \wedge \ X \cap O(t) = \emptyset$ and then its occurrence leads to the marking $M' = X \uplus O(t)$. Clearly, one can see the class of elementary nets as a special subclass of p/t nets with capacity, where the capacity of each place is equal to 1 and values of the input and output function $I(t)(p), O(t)(p) \in \{0, 1\}$ for every $t \in T$ and $p \in P$.

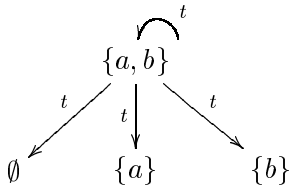
2.2 Extensions

In this section we deal with some abstract generalisations of p/t nets found in the literature. One of the first successful abstractions of p/t nets can be found in the paper [MM90], based on the idea suggested in [Win87].

There the theory is built over definition 2.1.1. For our purpose the paper is important by the fact that, to our best knowledge, it first proposes a generalisation of the p/t net algebra. As possible algebra there are suggested commutative monoids (the category of p/t nets with such assumption is called ***GralPetri*** in [MM90]), and generally semimodules.

According to [MM90], for objects of ***GralPetri*** we have that markings are elements of a commutative monoid $(E, +)$, *i.e.* $M \in E$ and functions I, O associate with each transition an element of E, *i.e.* $I, O : T \to E$. Then we have that a transition $t \in T$ is enabled to occur in a $M \in E$ iff there exists $X \in E$ such that $X + I(t) = M$, and its occurrence leads to the marking M' such that $M' = X + O(t)$.

Example 2.2.1 Given a set $P = \{a, b\}$, take for the domain of markings noncancellative commutative monoid formed by the power set of P with standard union, *i.e.* $(2^P, \cup)$. Let $T = \{t\}, I(t) = \{a, b\}, O(t) = \emptyset$, and $M_0 = \{a, b\}$. Then we have that occurrence of t in M leads to M' such that $M' \cup \{a\} = M$. From the following figure with the sequential case sequential graphs of such a net we can see that occurrence of t is non-deterministic, *e.g.* in $\{a, b\}$ it may lead again to $\{a, b\}$, but also to $\emptyset$, $\{a\}$ or $\{b\}$.

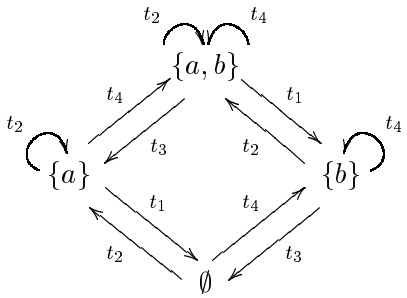


The idea given in [MM90] is further generalised in [EPR94] and [Pad96]. The algebra considered is generally a commutative semigroup. So, according to [Pad96] abstract Petri nets are given an follows:

Definition 2.2.2 *A net structure functor is a composition $G \circ F$, where F is a functor from the category of sets to the category of commutative semigroups and G is his right adjoint. Given a net structure functor $G \circ F$, a low level abstract Petri net is a tuple (P, T, I, O), where P and T are sets of places and transitions, respectively, and I, O are functions from T to $G \circ F(P)$.*

A marking of a low-level abstract Petri net is an element $M \in F(P)$. Because of the properties of adjunction, for each $f : T \to G \circ F(P)$ there exists a unique extension $\bar{f} : F(T) \to F(P)$. Then the enabling rule and occurrence of "transition vectors" are defined as follows: given a "transition vector" $v \in F(T)$, it is said that v is enabled to occur in M iff there exists $X \in F(P)$ such that $X + \bar{I}(v) = M$, and then the occurrence of v leads to the new marking M' given by $M' = X + \bar{O}(v)$. [Pad96] mentions the possibility of non-determinism caused by non-unique solutions of the equation $X + \bar{I}(v) = M$. It is solved by specifying additional conditions.

Example 2.2.3 Recall, that in [Pad96] a net, where $F(P) = \mathcal{P}(P)$ is the standard powerset functor mapping a set to its power set with union, is called unsafe elementary net. The problem of non-unique solution of the equation $X \cup \bar{I}(v) = M$ is solved by demanding distinct union in the equation.
Now, take again as in the previous example $P = \{a, b\}$, and an algebra of the net $(2^P, \cup)$. We have for a marking M that $M \in 2^P$, *i.e.* $M \in \mathcal{P}(P)$. Let us consider that we use only distinct union in equation $X \cup I(t) = M$, so we have that a transition t is enabled to occur in M iff exists $X \in 2^P$ such that $X \uplus I(t) = M$. One may easily rewrite this for a $v \in \mathcal{P}(T)$. Let $T = \{t_1, t_2, t_3, t_4\}$, let $I(t_1) = \{a\}, I(t_3) = \{b\}, O(t_2) = \{a\}, O(t_4) = \{b\}$ and let every other value of input and output function be equal to empty set. Using the terminology from [Pad96], we have an unsafe elementary net. So, we have that the occurrence of a transition in M leads to M' such that for $t_1 : M' \uplus \{a\} = M$; for $t_2 : M' = M \cup \{a\}$; for $t_3 : M' \uplus \{b\} = M$; for $t_4 : M' = M \cup \{b\}$;



As we can see from the previous figure with the sequential case graph of the net, although demanding in the equation $X \uplus I(t) = M$ distinct union in order to remove non-determinism, we have that, for example, the sequence of transitions $t_2 t_1 t_3$ changes the marking $\{a, b\}$ to the

marking $\emptyset$, but the occurrence of the sequence of transitions $t_1t_3t_2$ change the same marking $\{a, b\}$ to the marking $\{a\}$. In the notions of "transition vectors" we have that the sequence of "vectors" vw, where $v = \{t_1, t_3\}$ and $w = \{t_2\}$, changes $\{a, b\}$ to $\emptyset$ but wv changes the {a,b} to the marking $\{a\}$. This generally means that *using noncancellative commutative monoids as Petri net algebra, change of the state can depend on the order of the transition occurrences,* i.e. *commutativity of transition occurrences (if enabled) is not satisfied!* However, the commutativity is just the property that permits overcoming sequentiality in Petri net computations!

One could note that strictly formally power set with distinct union is a *partial* commutative semigroup, although in definition 2.2.2 according to [Pad96] a full commutative semigroup is required. Also generally, taking a noncancellative commutative monoid, the computation through sequences of firing vectors may differ from the computation through the composition of 'firing vectors', *e.g.* in the case of example 2.2.3 (when the net functor associates the set of transition with its powerset with union) the sequence of "vectors" vw, where $v = \{t_1, t_3, t_2\}$ and $w = \{t_1\}$ changes $\{a, b\}$ to $\emptyset$ but $v \cup w = v$ changes the $\{a, b\}$ to the marking $\{a\}$.

3 Extension of p/t nets, transition systems and Abelian groups

Based on the extensions discussed in the previous section, in this section we suggest a very general algebraic structure of nets for the purpose of investigating different extensions and their relationship. Then we choose a tuple of properties that holds in the sequential case graph of each standard p/t net. Further we show which kind of (partial) algebra, if it is used in p/t net instead of integers with addition, preserves (is equivalent to) this tuple of properties. In this general definition we do not pay attention to distributivity of p/t net, only to the algebra used.

In the field of Petri nets, as we can see from the example of widely used elementary nets where distinct union is used, one may use a partial binary algebra, *i.e.* a partial groupoid. In order to cover also these cases, we will generally allow partiality of composition. As follows we recall very basic definitions and notation from theory of partial groupoids (according to [LE91]) used through the paper.

3.1 Partial binary algebra

Definition 3.1.1 *A partial groupoid is an ordered tuple $\mathcal{H} = (H, \perp, \dot{+})$ where H is a carrier of $\mathcal{H}$, $\perp \subseteq H \times H$ is the domain of $\dot{+}$, and $\dot{+} : \perp \to H$ is a partial operation of $\mathcal{H}$.*

Definition 3.1.2 *We say that a partial groupoid $\mathcal{H} = (H, \perp, \dot{+})$ can be embedded (is embeddable) to an Abelian group iff there exists an Abelian group $(G, +)$ such that $H \subseteq G$ and the*

operation $+$ *restricted on* $\perp$ *is equal to the partial operation* $\dot{+}$*, in symbols* $+|_{\perp} = \dot{+}$*. Group* $(G, +)$ *is called embedding of partial groupoid* $\mathcal{H}$*.*

Recall that a total groupoid is embeddable to an Abelian group if and only if it is a cancellative commutative associative groupoid, *i.e.* a cancellative commutative semigroup. Remember that a left cancellative semigroup is a semigroup where $\forall a, b, c \in H : a + b = a + c \Longrightarrow b = c$. In similar way is defined right cancellative semigroup. A semigroup is said to be cancellative if it is both left and right cancellative. Clearly for a commutative semigroup left and right cancellativity coincides. For more details about embedding of semigroups to groups see *e.g.* [CP67]. In the case of proper partial groupoid (*i.e.* $\perp \subset H \times H$), associativity, commutativity and cancellativity are only necessary conditions of embeddability to an Abelian group, but not sufficient.

3.2 Let's start

First we define a very general net state functor. Let us denote the category of sets by **SET** and the category of partial groupoids by **PGROUPOID**.

Further, let U : **PGROUPOID** $\rightarrow$ **SET** be the forgetful functor, *i.e.* given any partial groupoid $\mathcal{H} = (H, \perp, \dot{+})$, we have $U(\mathcal{H}) = H$.

Definition 3.2.1 *A net state functor is a functor* F : **SET** $\rightarrow$ **PGROUPOID** *associating a set with a partial groupoid.*

Definition 3.2.2 *Given a net state functor* F*, an algebraically generalised place/transition net is an ordered tuple* $\mathcal{AN} = (P, T, I, O)$*, where* P *and* T *are distinct sets of places and transitions; and* $I, O : T \rightarrow U \circ F(P)$ *are input and output functions, respectively.*

The structure $F(P)$ is called (partial) algebra of $\mathcal{AN}$. As follows, let $F(P) = (H, \perp, \dot{+})$. A state of net $\mathcal{AN}$, also called marking or case, is an element $M \in H = U \circ F(P)$. A transition $t \in T$ is enabled to occur in a state $M \in H$ if and only if $\exists X \in H$ such that $X \perp I(t) \ \wedge \ X \dot{+} I(t) = M \ \wedge \ X \perp O(t)$, and then its occurrence leads to the marking $M' = X \dot{+} O(t)$.

Definition 3.2.3 *As usual, a* marked *algebraically generalised p/t net is a pair* $(\mathcal{AN}, M_0)$*, where* $\mathcal{AN}$ *is an algebraically generalised p/t net and* M_0 *is a distinguished initial state of the net* $\mathcal{AN}$*.*

Definition 3.2.4 *Similarly to standard p/t nets,* sequential case graph *of an algebraically generalised p/t net* $\mathcal{AN} = (P, T, I, O)$ *with partial algebra F(P) is the transition system* $(U \circ F(P), T, \Leftrightarrow\!\!\rightarrow)$*, where*

$$M \overset{t}{\Leftrightarrow\!\!\rightarrow} M' \Leftrightarrow \exists X \in U \circ F(P) : X \perp I(t) \wedge X \dot{+} I(t) = M \wedge X \perp O(t) \wedge M' = X \dot{+} O(t).$$

Then, given an initial state $M_0 \in U \circ F(P)$, the sequential case graph of marked algebraically generalised p/t net $(\mathcal{AN}, M_0)$ is the pointed labelled transition system $(\{M_0 \Leftrightarrow\!\!\!\rightarrow_\star\}, T, \Leftrightarrow\!\!\!\rightarrow \cap \{M_0 \Leftrightarrow\!\!\!\rightarrow_\star\} \times T \times \{M_0 \Leftrightarrow\!\!\!\rightarrow_\star\}, M_0)$.

It is evident that standard p/t nets from definition 2.1.1 are algebraically generalised nets with the state functor associating the set of places P with the free commutative monoid over P. Elementary nets from definition 2.1.6 are algebraically generalised p/t nets with the state functor associating a set of places with the partial groupoid of its powerset with distinct union. Finally, a p/t net with capacity $\mathcal{KN} = (P, T, I, O, K)$ (see def. 2.1.2) is a net with $F(P) = (\mathbb{N}^P, \perp = \{(X, Y) \mid \forall p \in P : X(p) + Y(p) \in \{0, \cdots, K(p)\}\}, \dot{+} = +|_\perp)$. Moreover, algebra of standard p/t nets, and partial algebra of elementary nets and generally p/t nets with capacity is embeddable to the free Abelian group $\mathbb{Z}^P$.

As follows, we choose a tuple of properties, that holds in sequential case graph of each standard (marked) p/t net given by definition 2.1.1. Then we find which kind of (partial) algebra used in algebraically generalised p/t nets instead of integers with addition preserves this tuple of properties.

Recall that given a sequence q over a set A we write b_q to denote Parikh's image of q, *i.e.* $b_q \in \mathbb{N}^A$ is a multi-set in which the number of the occurrences $b_q(a)$ of each element a from A is given by the number of its occurrences in q.

Definition 3.2.5 *Let $\mathcal{S} = (S, L, \Leftrightarrow\!\!\!\rightarrow)$ be a labelled transition system. We say that system $\mathcal{S}$ is commutative iff $\forall\, h \overset{q}{\Leftrightarrow\!\!\!\rightarrow}_\star s, h \overset{q'}{\Leftrightarrow\!\!\!\rightarrow}_\star s' : b_q = b_{q'} \Rightarrow s = s'$.*

Evidently, the sequential case graph of the net from example 2.2.3 is not a commutative transition system. It is also clear that every commutative labelled transition system is deterministic.

For commutative labelled transition systems it is straightforward to extend the string transition relation $\Leftrightarrow\!\!\!\rightarrow_\star$ to *multi-set transition relation* $\Leftrightarrow\!\!\!\rightarrow_\diamond \subseteq S \times \mathbb{N}^L_{fin} \times S$ such that $(s, b, s') \in \Leftrightarrow\!\!\!\rightarrow_\diamond$ iff there exists $q \in L^*$ such that $s \overset{q}{\Leftrightarrow\!\!\!\rightarrow}_\star s' \wedge b_q = b$. As usual, we write $s \overset{b}{\Leftrightarrow\!\!\!\rightarrow}_\diamond s'$ to denote $(s, b, s') \in \Leftrightarrow\!\!\!\rightarrow_\diamond$ and $s \overset{b}{\Leftrightarrow\!\!\!\rightarrow}_\diamond$ to denote that $\exists s' \in S : s \overset{b}{\Leftrightarrow\!\!\!\rightarrow}_\diamond s'$. Clearly, $s \overset{b}{\Leftrightarrow\!\!\!\rightarrow}_\diamond s' \overset{b'}{\Leftrightarrow\!\!\!\rightarrow}_\diamond s'' \Longrightarrow s \overset{b+b'}{\Leftrightarrow\!\!\!\rightarrow}_\diamond s''$.

Remark 3.2.6 Recall that a congruence $\approx$ on an Abelian group $(G, +)$ is an equivalence relation on G preserving the group operation, *i.e.* $a \approx b \ \wedge \ c \approx d \Longrightarrow (a + c) \approx (b + d)$ for every $a, b, c, d \in G$. As usual, given an element $g \in G$, we denote by $[g]_\approx = \{g' \mid g' \approx g\}$ the equivalence class containing the element g, and by $+_{/\approx}$ the operation on $G/_\approx$ given as follows: $\forall [g]_\approx, [g']_\approx \in G/_\approx : [g]_\approx +_{/\approx} [g']_\approx = [g + g']_\approx$. In other words, $(G/_\approx, +_{/\approx})$ is the factor group of the group $(G, +)$ according to the congruence $\approx$. For a more general statement of congruence we refer to [Adá83].

Definition 3.2.7 *Given a commutative labelled transition system $\mathcal{S} = (S, L, \Leftrightarrow)$, let relation $\sim_{\mathcal{S}} \subseteq \mathbb{N}^L_{fin} \times \mathbb{N}^L_{fin}$ be such that $b \sim_{\mathcal{S}} b' \iff \exists\ s \overset{b}{\Leftrightarrow}_\diamond s', s \overset{b'}{\Leftrightarrow}_\diamond s'$. Let $\approx_{\mathcal{S}} \subseteq \mathbb{Z}^L_{fin} \times \mathbb{Z}^L_{fin}$ be the least congruence on $\mathbb{Z}^L_{fin}$ containing $\sim_{\mathcal{S}}$,* i.e. *$\sim_{\mathcal{S}} \subseteq \approx_{\mathcal{S}}$. We say that $\mathcal{S}$ is consensual iff $\forall\ h \overset{b}{\Leftrightarrow}_\diamond s, h \overset{b'}{\Leftrightarrow}_\diamond s' : b \approx_{\mathcal{S}} b' \Rightarrow s = s'$.*

As follows, we simply write $\sim$ and $\approx$ to denote $\sim_{\mathcal{S}}$ and $\approx_{\mathcal{S}}$ if system $\mathcal{S}$ is clear from the context.

Remark 3.2.8 We choose the name 'consensual' because it expresses a kind of 'common opinion' of the multi-sets on the problem "how to change the state". If they change a state in the same way once (*i.e.* from common source state to common target state) then they do it also for all other states (*i.e.* if they are in common source state and both can occur, then they lead to the common target state again). Moreover, this 'consensus' is transitive and closed under addition and subtraction of multi-sets.

Lemma 3.2.9 *The sequential case graph $(S, L, \Leftrightarrow)$ of every standard place/transition net $\mathcal{N} = (P, T, I, O)$ (given by def. 2.1.1) is commutative and consensual.*

Proof

For the sequential case graph of a p/t net $\mathcal{N}$ we have from its definition: $M \overset{q}{\Leftrightarrow}_\star M' \Longrightarrow M' = M + \hat{C}(b_q)$ for every $M \overset{q}{\Leftrightarrow}_\star M'$.

Commutativity: take any $M \overset{q}{\Leftrightarrow}_\star M', M \overset{q'}{\Leftrightarrow}_\star M''$ such that $b_q = b_{q'}$. From previous we have directly that $M' = M + \hat{C}(b_q) = M + \hat{C}(b_{q'}) = M''$ $\square$.

Now we show consensuality, *i.e.* we show that $\forall M \overset{b}{\Leftrightarrow}_\diamond M', M \overset{b'}{\Leftrightarrow}_\diamond M'' : b \approx b' \Rightarrow M' = M''$. Take $\cong \subseteq \mathbb{Z}^T_{fin} \times \mathbb{Z}^T_{fin}$ such that $b \cong b' \iff \hat{C}(b) = \hat{C}(b')$. One can easily check, that $\cong$ is a congruence on Abelian group $\mathbb{Z}^T_{fin}$ (clearly, it is an equivalence, and given any $\hat{C}(b) = \hat{C}(b')$ and $\hat{C}(d) = \hat{C}(d')$ we have $\hat{C}(b) + \hat{C}(b') = \hat{C}(d) + \hat{C}(d')$, and further $\hat{C}(b) + \hat{C}(b') = \sum_{t \in T} C(t) \cdot b(t) + \sum_{t \in T} C(t) \cdot b'(t) = \sum_{t \in T} C(t) \cdot (b(t) + b'(t)) = \hat{C}(b + b')$ and the same for d, d', *i.e.* we have $\hat{C}(b + b') = \hat{C}(d + d')$).

Given any $M \overset{b}{\Leftrightarrow}_\diamond M', M \overset{b'}{\Leftrightarrow}_\diamond M''$ we have that if $b \cong b$, (*i.e.* $\hat{C}(b) = \hat{C}(b')$ and therefore $M' = M + \hat{C}(b) = M + \hat{C}(b') = M''$) then $M' = M''$.

Because $\approx$ is the least congruence on $\mathbb{Z}^T_{fin}$ containing $\sim$, now it suffices to show that $\sim \subseteq \cong$.

For every $b \sim b'$ we have from the definition of $\sim$ that $\exists M \overset{b}{\Leftrightarrow}_\diamond M', M \overset{b'}{\Leftrightarrow}_\diamond M'$, *i.e.* $M' = M + \hat{C}(b) = M + \hat{C}(b') \Longrightarrow \hat{C}(b) = \hat{C}(b')$, *i.e.* $b \cong b'$ $\square$.

Lemma 3.2.10 *Let $\mathcal{S} = (S, L, \Leftrightarrow)$ be a reachable labelled transition system. Then $\mathcal{S}$ is commutative and consensual if and only if there exists an Abelian group $\mathcal{G} = (G, +)$ that holds: $\exists$ an injection $\sigma : S \to G\ \wedge\ \exists f : L \to G$ such that $\forall\ s \overset{a}{\Leftrightarrow} s' : \sigma(s) + f(a) = \sigma(s')$.*

Proof
$\Longrightarrow$ Because $\approx$ is a congruence on Abelian group $\mathbb{Z}^L_{fin}$, also $\mathcal{G} = (\mathbb{Z}^L_{fin}/_\approx, +_{/\approx})$ is an Abelian group (it is a factor group of $\mathbb{Z}^L_{fin}$ with respect to $\approx$, so we have $\forall\ b, b' \in \mathbb{Z}^L_{fin} : [b + b']_\approx = [b]_\approx +_{/\approx} [b']_\approx$). System $\mathcal{S}$ is reachable, *i.e.* we have a state, say $r \in S$, from which each $s \in S$ is reachable. Now, let σ be defined as follows: $\sigma(r) = [0]_\approx$ and $\forall s \in S : \sigma(s) = [b]_\approx$ where $r \overset{b}{\Leftrightarrow}_\diamond s$. One can check that σ is well defined (given any two b, b' such that $r \overset{b}{\Leftrightarrow}_\diamond s$ as well as $r \overset{b'}{\Leftrightarrow}_\diamond s$ there is $b \approx b'$ and therefore $[b]_\approx = [b']_\approx$), and it is an injection ($\sigma(s) = \sigma(s')$ means that there exists $r \overset{b}{\Leftrightarrow}_\diamond s$ and $r \overset{b'}{\Leftrightarrow}_\diamond s'$ such that $[b]_\approx = [b']_\approx$, *i.e.* $b \approx b'$, which, because $\mathcal{S}$ is consensual, implies $s = s'$).
Let $f : L \to \mathbb{Z}^L_{fin}/_\approx$ be given by $f(a) = [b_a]_\approx$ for every $a \in L$.
Now take arbitrary $s \overset{a}{\Leftrightarrow} s'$, *i.e.* $s \overset{b_a}{\Leftrightarrow}_\diamond s'$. We have that $\sigma(s) = [b]_\approx$, where $r \overset{b}{\Leftrightarrow}_\diamond s$. But then we also have $r \overset{b+b_a}{\Leftrightarrow}_\diamond s'$ and therefore $\sigma(s') = [b + b_a]_\approx = [b]_\approx +_{/\approx} [b_a]_\approx = \sigma(s) +_{/\approx} f(a)$.
$\Longleftarrow$ as in lemma 3.2.9.
□.

We showed that in a reachable transition system commutativity and consensuality is equivalent to the existence of an Abelian group playing the role of algebra in the system.

As follows a system in which computations may be expressed using elements and the composition law of an Abelian group is called Abelian group transition system.

Definition 3.2.11 *A labelled Abelian group transition system is such a labelled transition system $\mathcal{S} = (S, L, \Leftrightarrow)$ that there exists an Abelian group $\mathcal{G} = (G, +)$ that holds: $\exists$ an injection $\sigma : S \to G\ \wedge\ \exists f : L \to G$ such that $\forall s \overset{a}{\Leftrightarrow} s' : \sigma(s) + f(a) = \sigma(s')$. The group $\mathcal{G} = (G, +)$ is said to be associated with system $\mathcal{S}$, the injection σ is called state injection, and the function f is called incidence function of $\mathcal{S}$. If also f is an injection then it is said that $\mathcal{S}$ is unambiguously labelled.*

For algebraically generalised p/t nets there hold the following claim:

Lemma 3.2.12 *A pointed transition system $\mathcal{PS} = (S, L, \Leftrightarrow, i)$ is a labelled Abelian group transition system if and only if it is isomorphic to the sequential case graph of a marked algebraically generalised p/t net $\mathcal{MAN} = (P, T, I, O, M_0)$ in which partial algebra F(P) can be embedded to an Abelian group.*

Proof
$\Longleftarrow$ Let $(G, +)$ denote an Abelian group to which the partial algebra $F(P)$ can be embedded. Then it suffices to take $f : T \to G$, $f(t) = O(t) \Leftrightarrow I(t)$ for all $t \in T$, to show that the sequential case graph of $\mathcal{MAN}$ is an Abelian group transition system. □

$\Longrightarrow$ We have an Abelian group $(G,+)$ with a state injection $\sigma : S \to G$ and an incidence function $f : L \to G$ such that $\forall s \stackrel{a}{\Leftrightarrow} s' : \sigma(s) + f(a) = \sigma(s')$. Without loss of generality, we can assume that $S \subseteq G \ \wedge \ \sigma(s) = s$ for every $s \in S$, and hence $\forall\ s \stackrel{a}{\Leftrightarrow} s' : s + f(a) = s'$. Let $(K,+)$ be an Abelian group such that $L \subseteq K$. Now denote by $(H,+)$ direct product of $(G,+)$ and $(K,+)$ (*i.e.* $(H,+) = (G,+) \oplus (K,+)$, and we write (g,k) to denote an element of H, and 0 both for the neutral element of G and K). For every $a \in L$ let relation $\perp_a = (\{s \mid s \stackrel{a}{\Leftrightarrow}\} \times \{\Leftrightarrow a\}) \times (\{f(a),0\} \times \{a\}) \subseteq H \times H$. Finally denote $\perp = \cup_{a \in L} \perp_a$. So we have a partial algebra $\mathcal{H} = (H, \perp, +|_\perp)$ which can be embedded to Abelian group $(H,+)$. As follows we write as usual $\dot{+}$ instead of $+|_\perp$.
Now take, for simplicity, the constant net state functor $F : \mathbf{SET} \to \mathbf{PGROUPOID}$ that associates with each set partial groupoid $\mathcal{H}$ (and with each function between sets the identity morphism $id : \mathcal{H} \to \mathcal{H}$).
Define an algebraically generalised net $\mathcal{AN} = (P,T,I,O)$, where P is an arbitrary set, $T = L$, and $I, O : T \to U \circ F(P)$ are given as follows:
$\forall a \in T : I(a) = (0,a) \wedge O(a) = (f(a),a)$
Let $(H,T,\rightarrowtail)$ be the sequential transition system of $\mathcal{AN}$. From definition of $\perp$ (only $(s, \Leftrightarrow a) \perp (I(a) = (0,a))$ and $(s, \Leftrightarrow a) \perp (O(a) = (f(a),a))$ where $s \stackrel{a}{\Leftrightarrow}$, are in the relation $\perp$) we have that $\forall\ a \in T\ \forall X \in H : X \perp I(a) \Longrightarrow (X \dot{+} I(a)) = (s,0) \in S \times \{0\}$ and also $X \perp O(a) \Longrightarrow (X \dot{+} O(a)) = (s',0) \in S \times \{0\}$. From the enabling and firing rule (recall that $a \in T$ is enabled in $M \in H$ iff exists $X \in H : X \perp I(a) \wedge X \dot{+} I(a) = M \wedge X \perp O(a)$, and then occurrence (firing) of a leads to $M' = X \dot{+} O(a)$), we have that $\rightarrowtail \subseteq (S \times \{0\}) \times T \times (S \times \{0\})$, or in other words $\rightarrowtail = \rightarrowtail \cap ((S \times \{0\}) \times T \times (S \times \{0\}))$. Because of the construction of $\perp$ and the enabling and firing rule, we further have that

$$\forall a \in T\ \forall M \in S \times \{0\} : M = (s,0) \stackrel{a}{\rightarrowtail} M' = (s',0) \iff s \stackrel{a}{\Leftrightarrow} s' \qquad (1)$$

From the definition of $\{(i,0) \rightarrowtail_\star\}$ we have that for every $h \in \{(i,0) \rightarrowtail_\star\}$ there exists a, possibly empty, string of transitions $q = a_1 \ldots a_n \in T^*$ such that $(i,0) \stackrel{a_1}{\rightarrowtail} h_1 \cdots \stackrel{a_n}{\rightarrowtail} h_n = h$. Because $\rightarrowtail \subseteq (S \times \{0\}) \times T \times (S \times \{0\})$, we have that $\{(i,0) \rightarrowtail_\star\} \subseteq S \times \{0\}$. On the other hand, because $\{i \Leftrightarrow_\star\} = S$, we have that for every $s \in S$ there exists a, possibbly empty, string of labels $q = a_1 \ldots a_n \in L^*$ such that $i \stackrel{a_1}{\Leftrightarrow} s_1 \cdots \stackrel{a_n}{\Leftrightarrow} s_n = s$. From (1) we further have that $(i,0) \stackrel{a_1}{\rightarrowtail} (s_1,0) \cdots \stackrel{a_n}{\rightarrowtail} (s_n,0) = (s,0)$, i.e. we have $S \times \{0\} \subset \{(i,0) \rightarrowtail_\star\}$. So, we get $\{(i,0) \rightarrowtail_\star\} = S \times \{0\}$. Directly from definition 3.2.4 we have that the pointed transition system $(S \times \{0\}, T, \rightarrowtail, (i,0))$ is the sequential case graph of the marked algebraically generalised p/t net $\mathcal{MAN} = (P,T,I,O,(i,0))$, in which partial algebra $F(P)$ is embeddable to an Abelain group. Clearly, there is an isomorphism (given by bijection $\beta : S \times \{0\} \to S$, $\beta(s,0) = s$ for each $s \in S$; and identity $T = L$) between the pointed transition system $(S, L, \Leftrightarrow, i)$ and the sequential case graph $(S \times \{0\}), T, \rightarrowtail, (i,0))$ of the marked net $\mathcal{MAN}$.
□

Remark 3.2.13 The importance of the group $(K,+)$ in the above proof is that it enables

to maintain the cases, where some transitions cause the same change but the domains on which they are enabled to occur differ. In particular, it allows to distinguish selfloops each others and also from 'no' action in our construction. The consideration of possibly different domains for transitions with the same effect gives real sense in the case of ambiguous labelling of transition systems.

Finally, we can formulate the following theorem for marked p/t nets that directly follows from lemma 3.2.10 and lemma 3.2.12.

Theorem 3.2.14 *A pointed transition system $\mathcal{S} = (S, L, \Leftrightarrow, i)$ is commutative and consensual if and only if it is isomorphic to the sequential case graph of a marked algebraically generalised p/t net $\mathcal{MAN}$ in which partial algebra $F(P)$ can be embedded to an Abelian group.*

We have shown properties corresponding to the use of partial algebra embeddable to an Abelian group in Petri nets. Someone can choose a more general algebra (as a commutative monoid, or something what can be embedded to a commutative monoid), but should be prepared, that by doing this some of these properties can be lost (in the case of commutative monoids, as we demonstrated on example 2.2.3, we might even lose commutativity!). This is our main message.

4 Conclusion

In the paper we have studied an extension of Petri nets based on generalising algebra used in the dynamics of nets. More precisely, we have studied the role of algebra used in Petri nets through properties of their interleaving semantic - labelled transition systems called traditionally sequential case graphs. Our motivation and aim have been to extend the class of systems that can be (directly) modelled by algebraically generalised Petri nets, more precisely to extend the class of labelled transition systems isomorphic to the sequential case graph of an algebraically generalised Petri net while preserving some natural semantic properties guaranteed in standard Petri nets.

We have started with a very general definition of Petri nets using a state functor mapping the set of places to partial groupoids, *i.e.* to Petri net (partial) algebra.

For purpose of the study we have chosen the pair of semantic properties preserved in sequential case graphs of all standard Petri nets, namely *commutativity* of occurrences of transitions, which enables to use multi-sets instead of sequences of transitions, and *consensuality* of computations in Petri nets (that means if the occurrences of two sequences of transitions in one state (marking) lead to the common new state, then they lead to the common new state from every other state in which both sequences are enabled to occur, and moreover, this 'consensus' is transitive and closed under addition and subtraction of multi-sets). These two

properties are crucial for possibility of using linear algebraic techniques in the description and analysis of Petri nets.

We have shown that using arbitrary commutative monoid as Petri net algebra, as suggested in [MM90], does not preserve determinism and commutativity of occurrences of transitions in sequential case graphs of Petri nets, and although one uses some additional requirements to satisfy determinism, as it is done in [Pad96], the sequential case graph still may not preserve commutativity. In other words, the construction of Petri nets causes that the commutativity of a monoid does not guarantee the commutativity of the corresponding sequential case graphs.

In this paper it is proven that for a pointed (reachable) labelled transition system $\mathcal{S}$ the following statements are equivalent:

- $\mathcal{S}$ is commutative and consensual;
- $\mathcal{S}$ is an Abelian group transition system, *i.e.* a system in which computation may be expressed using elements and the composition law of an Abelian group;
- $\mathcal{S}$ is (isomorphic to) the sequential case graph of a marked algebraically generalised Petri net the (partial) algebra of which is embeddable to an Abelian group.

Acknowledgement

I would like to thank Mogens Nielsen for many valuable discussions, and Zhe Yang for his extensive comments, very valuable technical suggestions and careful reading of the manuscript. I also wish to thank Martin Drozda for reading and improvement of the draft of the paper. I am also grateful to Petr Jančar and Jörg Desel for their encouragement to continue on this work.

References

[Adá83] J. Adámek. *Theory of Mathematical Structures.* D. Reidel Publishing Company, Kluwer Academic, Dordrecht, Holland, 1983.

[CP67] A. H. Clifford and G. B. Preston. *The Algebraic Theory of Semigroups I, II.* American Mathematical Society, 1961, 1967.

[EPR94] H. Ehrig, J. Padberg, and G. Rozenberg. Behaviour and relization construction for Petri nets based on free monoid and power set graphs. In *Workshop on Concurrency, Specification & Programming.* Humboldt University, 1994. Extended version appeared as Technical Report of University of Leiden.

[JK91] R. Janicki and M. Koutny. Invariant semantics of nets with inhibitor arcs. In *Proceedings of CONCUR*, volume 527 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1991.

[JR91] K. Jensen and G. Rozenberg, editors. *High-Level Petri-Nets, Theory and Applications*. Springer-Verlag, Berlin, 1991.

[Kel76] R. M. Keller. Formal verification of parallel programs. *Communications of the ACM*, 7(19):371–384, 1976.

[KM69] Richard M. Karp and Raymond E. Miller. Parallel program schemata. *Journal of Computer and System Sciences*, 3(2):147–195, May 1969.

[LE91] E. S. Ljapin and A. E. Evseev. *Partial algebraic operations (in Russian)*. Izdatelstvo "Obrazovani", St. Petersburg, 1991.

[MM90] José Meseguer and Ugo Montanari. Petri nets are monoids. *Information and Computation*, 88(2):105–155, October 1990.

[Pad96] J. Padberg. *Abstract Petri Nets: Uniform Approach and Rule-Based Refinement*. PhD thesis, Technical University of Berlin, 1996.

[Pet62] C. A. Petri. *Kommunikation mit Automaten*. PhD thesis, University of Bonn, Bonn, West Germany, 1962.

[Pet73] J. L. Peterson. *Modeling of Parallel Systems*. PhD thesis, Department of Electrical Engineering, Stanford University, Stanford, CA, 1973.

[Pet81] J. L. Peterson. *Petri Net Theory and the Modeling of Systems*. Pentice-Hall, Englewood Cliffs, NJ, 1981.

[Win87] G. Winskel. Petri nets, algebras, morphisms, and compositionality. *Information and Computation*, 72:193–238, 1987.

[WN95] G. Winskel and M. Nielsen. Models for concurrency. In S. Abramsky, D. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 4, pages 1–148. Oxford University Press, 1995. Also appeared in BRICS Report Series RS-94-12.

Intensional Approaches for Symbolic Methods *

Olga Kushnarenko and Sophie Pinchinat

EP-ATR Group, IRISA, Univ. Rennes I, Campus de Beaulieu, 35042 Rennes Cedex France
email: {okouchna, pinchina}@irisa.fr

Abstract. We present a behavioral model for discrete event systems based on an intentional formalism, as a possible approach within the broader trend towards rich symbolic representations in verification. We define *Intentional Labeled Transition Systems* with associated combinators of parallel composition and event hiding, and we propose *symbolic bisimulation* to handle strong bisimulation intentionally. Further on, we explain how the methodology has been developed for the synchronous language SIGNAL, via the verification tool SIGALI.

Keywords: intentional transition systems, polynomials, (symbolic) bisimulation, synchronous languages, equivalence checking.

1 Introduction

Dynamic systems are systems that evolve according to their environment. In general, an evolution of the system, in a given state, depends on an input event (some information given by the environment); this evolution leads to some instantaneous output event and to state changes.

Synchronous languages have been designed to ease the programmer's task when dealing with such systems; they provide some primitives for concurrency and communication. They can be of different kinds. The most popular ones that have been designed in France are: ESTEREL [BC84] an imperative language, LUSTRE [Pla88] and SIGNAL [BLJ91] based on declarative approach. These languages naturally bear a semantics in terms of discrete event systems, and their control part concerns boolean valued signals. The synchronous features allow one to express synchronization constraints between the different (output and internal) events of the system and the input events of its environment. Hence, any operational semantics of such systems leads to automata labeled combinations of atomic events.

The automata semantics can then be used as a basis for the verification of SIGNAL programs. For classic temporal logics specification verifications, the tool SIGALI [DLB97] was developed; this tool is based on an intensional representation of the automata. Whereas SIGNAL programs equivalence checking was made extensionally by feeding other verification tools, e.g. such as ALDEBARAN [Fer84] or FCTOOLS [BRRD96], with the extensional description of the automata. Obviously, the size of the generated transition systems limits the extensional methods.

* Work supported by the Esprit SYRF Project 22703.

In this paper, we propose an intensional formalism based on the algebraic theory of polynomials for bisimulation checking which perfectly fits the spirit of the tool SIGALI. The "polynomial language" provides the programmer with an intermediate language to describe symbolic algorithms, in an intensional way, without bothering with the underlying implementation.

In our modelization approach, instead of considering extensionally all possible events for a given state change, we develop a formalism where actions of the automata are polynomials, these automata will be called *intensionally Labeled Transition Systems* (or *iLTS* for short). These polynomials are based on several variables (one for each atomic event) with coefficients in $\mathbf{Z}_3$, according to the following encoding: an atomic boolean event can either be *absent*, then encoded by 0, or *present* and equal to *true*, encoded by 1, or *present* and equal to *false*, encoded by -1. The solutions of a polynomial are composed events. iLTS naturally possess an interpretation in terms of classical labeled transition systems, but they permit to avoid the transition enumeration one would get by describing extensionally each event and transition. Moreover, the algebraic theory of polynomials offers simple definitions for *parallel composition* and *event hiding*, both combinators widely used to design complex systems.

The paper is organized as follows: Sections 2.1 and 2.2 introduce the intensional models and the combinators. Further on, Section 2.3, we propose a behavioral equivalence, called *symbolic bisimulation* over iLTS with good properties; it has the congruence property w.r.t. the combinators (see Theorem 6). This definition enables one to handle classic strong bisimulation in an intensional style (see Theorem 8). Then, in order to proceed to symbolic verification, Section 2.4 introduces a still more intensional semantics for systems: polynomial formalism is extended to describe the whole system, that is, all its legal transitions. The resulting models are called *Intensional Labeled Transition Systems* (ILTS). Section 3 explains how the developed theory is currently applied to the language SIGNAL: the options of the compiler plugged with the basic functions of the verification tool SIGALI [DLB97] allow us to perform polynomial handling for bisimulation computation. For lack of space, we refer to [KP98] for the proofs details.

2 Intensional Labeled Transition Systems

This section introduces *intensionally labeled transition systems*, *parallel composition* and *event hiding*, as well as the *symbolic bisimulation* behavioral equivalence. Then *intensional labeled transition systems* are proposed as a more intensional description for labeled transition systems, and an algorithm for the symbolic bisimulation computation is given.

In the following, we write $\mathbf{Z}_3$ for the finite field $\{-1, 0, 1\}$ in which $x^3 = x$ and $3x = 0$ for all $x \in \mathbf{Z}_3$. Let $\bar{Z}$ be a finite set of m distinct variables $Z_1, ..., Z_m$. We denote by $\mathbf{Z}_3[\bar{Z}]$ (or $\mathbf{Z}_3[Z_1, ..., Z_m]$) the set of polynomials over variables $Z_1, ..., Z_m$ which coefficients

range over $\mathbf{Z}_3$ with typical elements $P(\bar{Z})$ (or P for short), $P_1(\bar{Z}), \ldots$. We recall that $(\mathbf{Z}_3[\bar{Z}], +, *)$ is a ring.

2.1 Intensionally labeled transition systems

Definition 1. (Intensionally Labeled Transition Systems (iLTS)) An *m-dimensional intensionally Labeled Transition System (or m-iLTS)* is a structure $T = (Q, \bar{Z}, \rightarrow)$, where
• Q is set of states,
• $\bar{Z}$ is a set of m variables $Z_1, ..., Z_m$, and
• $\rightarrow \subseteq Q \times \mathbf{Z}_3[\bar{Z}] \times Q$. Each transition is labeled by a polynomial over the set $\bar{Z}$.

We write $q \stackrel{P(\bar{Z})}{\rightarrow} q'$ (or simply $q \stackrel{P}{\rightarrow} q'$), instead of $(q, P(\bar{Z}), q') \in \rightarrow$.
Given a polynomial $P(\bar{Z}) \in \mathbf{Z}_3[\bar{Z}]$, we associate its set of solutions $Sol(P) \subseteq \mathbf{Z}_3^m$, defined by $\{(z_1, ..., z_m) \in \mathbf{Z}_3^m | P(z_1, ..., z_m) = 0\}$. Then, iLTS can be understood as an "intensional" representation of classical labeled transition systems, where the labels are tuples in $\mathbf{Z}_3^m$: each arrow of the iLTS labeled by $P(\bar{Z})$ intensionally represents as many arrows labeled by some $\bar{z}$ where $\bar{z} \in Sol(P(\bar{Z}))$. We call $Ext(T)$ the corresponding "extensional" labeled transition system.

Now, it is worthwhile noting that in $\mathbf{Z}_3[\bar{Z}]$, polynomials $Z_1^3 - Z_1, ..., Z_m^3 - Z_m$ evaluate to zero. Then for any $P(\bar{Z}) \in \mathbf{Z}_3[\bar{Z}]$, one for instance has $Sol(P) = Sol(P + (Z_1^3 - Z_1))$, but also, $Sol(P) = Sol(-P) = Sol(P^2)$, etc... A very natural abstraction would be to consider iLTS modulo isomorphism, of course, but also modulo $\equiv$-equivalence over labels, where $P_1 \equiv P_2$ whenever $Sol(P_1) = Sol(P_2)$[1].
Fortunately, for algorithmic purposes, [Dut92] showed how to define a unique representative of each $\equiv$-equivalence class, called the *canonical generator*. This polynomial is the characteristic function of $Sol(P)$ and has at most degree 2.

Lemma 2. [Dut92] *Given a polynomial $P \in \mathbf{Z}_3[\bar{Z}]$, the canonical generator of $[P]_{\equiv}$ is computable.*

2.2 Operations over iLTS

The class of iTLS can be provided with the usual operations over (extensional) transition systems. Among them, the *parallel composition* and the *events hiding* play an important role in the complex systems design.

Parallel composition over iLTS imposes the compatibility of values between common events of the composed systems. From the extensional point of view, Definition 3 is the classical *synchronous parallel composition* as defined in ESTEREL, SIGNAL or LUSTRE languages, but the intensional approach avoids a part of the potential combinatorial explosion to compute the synchronized transitions.

[1] Also, we can consider the quotient ring $\mathbf{Z}_3(\bar{Z}) \stackrel{\text{def}}{=} \mathbf{Z}_3[\bar{Z}] / < Z_1^3 - Z_1, ..., Z_m^3 - Z_m >$, which is isomorphic to the ring of functions from $\mathbf{Z}_3^m$ in $\mathbf{Z}_3$ in order to restrict to polynomials of degree at most 2.

Definition 3. (Parallel composition of iLTS) Let $T_1 = (Q_1, \bar{Z}, \rightarrow_1)$ be an m_1-iLTS and $T_2 = (Q_2, \bar{U}, \rightarrow_2)$ be an m_2-iLTS with possible common variables between $\bar{Z}$ and $\bar{U}$. The *parallel composition* of T_1 and T_2, written $T_1 \mid T_2$, is $(Q_1 \times Q_2, \bar{Z} \cup \bar{U}, \rightarrow)$ with

$$(q_1, q_2) \overset{P_1(\bar{Z}) \sqcap P_2(\bar{U})}{\rightarrow} (q_1', q_2') \text{ where } P_1 \sqcap P_2 \overset{\text{def}}{=} P_1^2 + P_2^2 \text{ whenever } q_1 \overset{P_1(\bar{Z})}{\rightarrow}_1 q_1' \text{ in } T_1 \text{ and } q_2 \overset{P_2(\bar{U})}{\rightarrow}_2 q_2' \text{ in } T_2.$$

Because in $\mathbf{Z}_3$, $P_1 \sqcap P_2 = 0$ iff $(P_1 = 0 \wedge P_2 = 0)$, we have $Sol(P_1 \sqcap P_2) = Sol(P_1) \cap Sol(P_2)$; it entails that $(P_1 \sqcap P_2) \sqcap P_3 \equiv P_1 \sqcap (P_2 \sqcap P_3)$. Therefore, parallel composition over iLTS is commutative and associative.

Hiding events consists in abstracting from components of the label. It helps in internalizing some communications between the composed systems that are not relevant to observe in the behavior.
Let $P \in \mathbf{Z}_3[\bar{Z}]$, we shall write $\exists Z_i P$ for the polynomial $P|_{Z_i=-1} * P|_{Z_i=0} * P|_{Z_i=1}$, where $P|_{Z_i=v}$ is P obtained by instantiating any occurrence of variable Z_i by value v. The reader can check that $Sol(\exists Z_i P)$ is obtained from $Sol(P)$ by deleting the i-th component of its elements (it is a projection). Also when $\tilde{Z} \subset \bar{Z}$ is some $\{Z_{i_1}, ..., Z_{i_r}\}$ we simply write $\exists \tilde{Z} P$ for $\exists Z_{i_1} ... \exists Z_{i_r} P$.

Also it is possible to define a dual variable abstraction over polynomials, based on universal quantificator: $\forall Z_i P$ is computed as $P|_{Z_i=-1} \sqcap P|_{Z_i=0} \sqcap P|_{Z_i=1}$ which solutions are elements of the form $(z_1, ..., z_{i-1}, z_{i+1}, ..., z_m)$ s.t. $\forall z_i, (z_1, ..., z_{i-1}, z_i, z_{i+1}, ..., z_m) \in Sol(P)$. This abstraction will be considered further on.

Definition 4. (Event hiding) Let $T = (Q, \bar{Z}, \rightarrow)$ be an m-iLTS, and $Z_i \in \bar{Z}$. We define the $(m-1)$-iLTS $(T \setminus \{Z_i\})$ by $(Q, \bar{Z} \setminus \{Z_i\}, \rightarrow_{\setminus\{Z_i\}})$ where $q_1 \overset{\exists Z_i P}{\rightarrow}_{\setminus\{Z_i\}} q_2$ iff $q_1 \overset{P}{\rightarrow} q_2$.

2.3 Symbolic bisimulation

As we aim to manipulate transition systems in an intensional way, we explain here how the classical strong bisimulation can be handle in this setting (see Theorem 8). The definition is strongly inspired from DeSimone's symbolic bisimulation over reactive automata [DR94].
In order to be compared, events of two iLTS have to belong to the same space $\mathbf{Z}_3^m$. This way, we suppose without of lost of the generality, two iLTS have the same events variables.

Definition 5. (Symbolic Bisimulation) Let $T_1 = (Q_1, \bar{Z}, \rightarrow_1)$ and $T_2 = (Q_2, \bar{Z}, \rightarrow_2)$ be two iLTS. A symbolic bisimulation between T_1 and T_2 is a binary relation $\mathcal{R} \subseteq Q_1 \times Q_2$ s.t. $q_1 \mathcal{R} q_2$ whenever
(1) for all $q_1 \overset{P}{\rightarrow}_1 q_1'$ there exists a finite set of transitions $(q_2 \overset{P_i}{\rightarrow}_2 q_2^i)_{i \in I}$ with
- $(1 - P^2) * \Pi_i P_i = 0$, which implies $Sol(P) \subseteq \bigcup_i Sol(P_i)$, and
- $q_1' \mathcal{R} q_2^i$, for all $i \in I$, and

(2) vice versa.

We have proved the congruence theorem for the symbolic bisimulation.

Theorem 6. (Compositionality) *Symbolic bisimulation is a congruence w.r.t. parallel composition and events hiding.*

We show here that symbolic bisimulation is an alternative view of strong bisimulation when intensional models are considered. At this stage, we assume the reader is familiar with the class of (extensional) labeled transition systems, as well as with the equivalence of *strong bisimulation*. However, we recall:

Definition 7. [Par81,Mil89] (Strong bisimulation) Given two transition systems (labeled over some set A) $t_1 = (Q_1, A, \rightarrow_1)$ and $t_2 = (Q_2, A, \rightarrow_2)$, a bisimulation between t_1 and t_2 is a binary relation $\rho \subseteq Q_1 \times Q_2$ s.t. $(q_1, q_2) \in \rho$ whenever
(1) for all $a \in A$, for all transition $q_1 \stackrel{a}{\rightarrow}_1 q_1'$ there exists a state q_2' s.t. $q_2 \stackrel{a}{\rightarrow}_2 q_2'$ and $(q_1', q_2') \in \rho$, and
(2) vice versa.

Symbolic bisimulation between iLTS corresponds to strong bisimulation between the extensional labeled transition systems:

Theorem 8. *Let T_1 and T_2 be two iLTS. Then there exists a symbolic bisimulation between T_1 and T_2 iff there exists a strong bisimulation between $Ext(T_1)$ and $Ext(T_2)$.*

2.4 Intensional Labeled Transition Systems

Intensional approach for labels offers a "compact" way to talk about sets of transitions in the system. However, we would like to reinforce this method in such a way that the whole system, and not only its sets of labels, can be itself described intensionally. For this purpose, a structure over the states is unavoidable. We propose the fairly standard structure of tuples for states where values ranges over booleans (in our setting it means values 1 and -1). This is classically used in symbolic verification methods.

Intuitively, the set of transitions will be given by a polynomial, which generalizes the iLTS approach. Applications in Section 3 will show how this formalism can be obtained for free from the real systems to be compared modulo symbolic (or equivalently, strong) bisimulation.

Definition 9. An (n, m)*-dimensional Intensional Labeled Transition System (or ILTS for short)* is a structure $S = (\bar{X}, \bar{Y}, \bar{Z}, \mathcal{T})$ where $\bar{X} = \{X_1, ..., X_n\}$ and $\bar{Y} = \{Y_1, ..., Y_n\}$ are two sets of *(source and target) states variables*, $\bar{Z} = \{Z_1, ..., Z_m\}$ is a set of labels variables and $\mathcal{T}(\bar{X}, \bar{Y}, \bar{Z})$ is a polynomial in $\mathbf{Z}_3[\bar{X}, \bar{Y}, \bar{Z}]$ describing the legal transitions.

Given some source state $\bar{x} = (x_1, ..., x_n) \in \mathbf{Z}_3^n$ and some target state $\bar{y} = (y_1, ..., y_n) \in \mathbf{Z}_3^n$, the set $Sol(\mathcal{T}(x, \bar{Z}, y))$ denotes all the possible labels of transitions from state $\bar{x}$ to state $\bar{y}$. When states are viewed extensionally, we retrieve the iLTS of in Section 2.1, which in

turn can be interpreted as a classical labeled transition system.

Now, an algorithm for computing the greatest strong bisimulation between two ILTS can be described as follows. Assume given two ILTS $S_1 = (\bar{X}^1, \bar{Y}^1, \bar{Z}, \mathcal{T}_1)$ and $S_2 = (\bar{X}^2, \bar{Y}^2, \bar{Z}, \mathcal{T}_2)$.

Algorithm

1. Define the polynomial $R_0(\bar{X}^1, \bar{X}^2) = 0$.
2. Compute iteratively until stabilization the sequence of polynomials $(R_k(\bar{X}^1, \bar{X}^2))_k$ defined by:

   $R_{k+1}(\bar{X}^1, \bar{X}^2)$ is the canonical generator of the $\equiv$-class of

$$\begin{cases} R_k(\bar{X}^1, \bar{X}^2) \\ \sqcap \ \forall \bar{Y}^1 \forall \bar{Z}[(1 - \mathcal{T}_1(\bar{X}^1, \bar{Y}^1, \bar{Z})^2) * \exists \bar{Y}^2 (\mathcal{T}_2(\bar{X}^2, \bar{Y}^2, \bar{Z}) \sqcap R_k(\bar{Y}^1, \bar{Y}^2))] \\ \sqcap \ \forall \bar{Y}^2 \forall \bar{Z}[(1 - \mathcal{T}_2(\bar{X}^2, \bar{Y}^2, \bar{Z})^2) * \exists \bar{Y}^1 (\mathcal{T}_1(\bar{X}^1, \bar{Y}^1, \bar{Z}) \sqcap R_k(\bar{Y}^1, \bar{Y}^2))] \end{cases} \quad (1)$$

Call $R(\bar{X}^1, \bar{X}^2)$ the result.

Theorem 10. (Termination and Correctness) *The algorithm terminates and at the end, $R(\bar{x}^1, \bar{x}^2) = 0$ iff there exists a bisimulation which relates states $\bar{x}_1$ and $\bar{x}_2$.*

Expression 1 can be made simpler when deterministic systems are to be compared. This is the case in Section 3, when our theory is applied to the synchronous language SIGNAL. Indeed, in this case the computation of R can be performed according to the following algorithm:

1. Compute the admissible events from a given state in each system: for system S_1, compute the canonical generator of $A_1(\bar{X}^1, \bar{Z})$ of $[\exists \bar{Y}^1 \mathcal{T}_1(\bar{X}^1, \bar{Y}^1, \bar{Z})]_{\equiv}$, and similarly compute $A_2(\bar{X}^2, \bar{Z})$ for S_2.
2. Compute the canonical generator $D_0(\bar{X}^1, \bar{X}^2)$ of $[\forall \bar{Z}(A_1(\bar{X}^1, \bar{Z}) - A_2(\bar{X}^2, \bar{Z}))]_{\equiv}$. Solutions of D_0 are pair of states $(\bar{x}^1, \bar{x}^2)$ which accept the same labels on their output transitions, i.e. which have the same *admissible* events.
3. Now the greatest invariant has to be computed. We iteratively compute polynomial D_k until stabilization as follows:

$$\begin{array}{l} D_{k+1}(\bar{X}^1, \bar{X}^2) \text{ is the canonical generator of the } \equiv\text{-class of} \\ \forall \bar{Y}^1 \forall \bar{Y}^2 \forall \bar{Z}[(1 - (\mathcal{T}_1(\bar{X}^1, \bar{Y}^1, \bar{Z}) \sqcap \mathcal{T}_2(\bar{X}^1, \bar{Y}^1, \bar{Z}))^2) * D_k(\bar{Y}^1, \bar{Y}^2)] \end{array} \quad (2)$$

3 Applications

The usual synchronous programs verification practice (in particular, the verification of *safety* properties [HLR93]) needs the use of parallel composition and event hiding operations. Since the parallel composition is synchronous, the desired properties of a

program can be easily and modularly expressed by means of an *observer*, i.e. another program which observes the behavior of the first one and decides whether it is correct. Then, the same formalism can be used to specify and to verify a complex system. The verification then consists in checking that the parallel composition of the two intensional transition systems never causes the observer to complain. It may happen that we just need a subset of signals: the property to verify can be expressed with this subset (for instance, the invariance under control property). It requires to specify the basic particular sets (of states and/or transitions) and to use event hiding. We need to make this handling easily available, so that program transformations remain internal and transparent, while powerful description is allowed.

As far as we are concerned, ILTS models are applied for the verification of systems described in the equational data-flow synchronous language SIGNAL [BLJ91][2]. This language is widely used to specify and to implement reactive systems as well as to verify their properties. There exists a lot of examples using the SIGNAL environment: among them, a production cell [ALGMR95], a power transformer station controller [LBMR96], an experiment with reactive data-flow tasking in active robot vision [RMC97], ...

The original multi-clock data-flow synchronous language SIGNAL manipulates a set of *signals*; each signal `A` denotes an unbounded series of typed values $(\mathtt{A}_t)_{t\in\mathcal{T}}$, indexed by time t in a time domain $\mathcal{T}$. $\perp$ is a particular value which denotes the absence of the signal. We call *clock of* `A` the set of instants t when `A` is not absent, i.e. $\mathtt{A}_t \neq \perp$. Two signals with the same clock are called *synchronous*. The *kernel*-language SIGNAL is based on four operations, defining primitive processes by equations, and a parallel composition to combine equations, as well as a signal hiding to internalize them.

In order to simplify the presentation, we shall restrict to the boolean fragment of SIGNAL language; that is the type domain is *true*, *false* or *absent*. The constructors of the language are equations of the form `A` :=< `expression` >, as well as a parallel composition and an event hiding.

• **Static synchronous operator** $\mathtt{A} := \mathtt{p}(\mathtt{A_1}, \ldots, \mathtt{A_n})$ is a boolean function of data $\mathtt{A_1}, \ldots, \mathtt{A_n}$ at each instant t. This instruction requires all referred variables to have the same clock.

• **Deterministic merge operator**, written `A` := `A1` `default` `A2`, `A` has the value of `A1` when `A1` is present, otherwise it has the value of `A2`. Its clock is the union of those of `A1` and `A2`.

• **Selection operator** of the form `A` := `A1` `when` `B` links `A` with `A1` when the boolean `B` has value *true*. The result can be seen as a down-sampling of a signal `A1`. The clock of `A` is the intersection of that of `A1` and the set of instants when boolean `B` has value *true*.

[2] developed in the EP-ATR research Group of the IRISA/INRIA Institute.

• **Delay** (a dynamic synchronous operator) `A` := `B` $1 gives access to the last value of signal `B`. `A` and `B` have equal clocks. The memorizing of last values will give raise to states (see below).
• **Parallel composition** of processes is noted | and consists in the conjunction of the equations (systems); it is then associative and commutative.
• **Signal hiding** \{`A`} hides any occurrence of signal `A`; it is internalized.

Logical SIGNAL programs can be translated into polynomials equations over $\mathbf{Z}_3$, following the principle of coding the possible values of a boolean signal `A` by a variable a: values for a will respectively be 1, -1 and 0 and are respectively interpreted by "`A` is *present* and *true*", "`A` is *present* and *false*", "`A` is *absent*"[3]. Therefore, any signal `A` can be associated its *clock* a^2, and two synchronous signals `A` and `B` satisfy $a^2 = b^2$.

Operators	Clock equations	Evaluations
`A` := `not` A_1	$a^2 = {a_1}^2$	$a = -a_1$
`A` := A_1 `and` A_2	$a^2 = {a_1}^2 = {a_2}^2$	$a = a_1 a_2 (a_1 a_2 - a_1 - a_2 - 1)$
		$a_1^2 = a_2^2$
`A` := A_1 `or` A_2	$a^2 = {a_1}^2 = {a_2}^2$	$a = a_1 a_2 (1 - a_1 a_2 - a_1 - a_2)$
		$a_1^2 = a_2^2$
`A` := A_1 `default` A_2	$a^2 = {a_1}^2 + (1 - {a_1}^2){a_2}^2$	$a = a_1 + (1 - {a_1}^2) a_2$
`A` := A_1 `when` `B`	$a^2 = {a_1}^2(-b - b^2)$	$a = a_1(-b - b^2)$
`A` := `B`$1	$a^2 = b^2$	$x' = b + (1 - b^2)x$
		$a = b^2 x$

Table 1. Synchronization constraints and the boolean signal evaluation

Table 1 shows how the programs are transformed into polynomial equations (we refer to [LBBLG91] for more details), leading to an ILTS models semantics. Nevertheless, the delay operator $ deserves some extra explanations. A delay requires to memorize the last value (then different from 0) of the signal into a (state) variable, say x. Translating `A` := `B` $1, imposes to introduce two auxiliary equations: (1) $x' = b + (1 - b^2)x$, where x' denotes the next value of state variable x, expresses the dynamics of the system. (2) $a = b^2 x$ delivers the value of the delayed signal according to the memorization in state variable x.

The translation of Table 1 is automatically performed by the SIGNAL compiler. The automata semantics can then be used as a basis for the verification of SIGNAL programs. To these ends, the tool SIGALI [DLB97], offering algebraic polynomial computations was developed. It relies on an implementation of polynomials by Ternary Decision Diagram (TDD) (for three valued logics) in the same spirit of BDD [Bry89], but where the paths in the data structures are decorated by values in $\{-1, 0, 1\}$ instead of $\{0, 1\}$. This tool performs classic temporal logics specification verifications, whereas until now, SIGNAL

[3] General SIGNAL programs, with other type values, can also be treated by only coding information of presence of absence of non-boolean signals.

programs equivalence checking was made extensionally: the tool SIGAUTO exports the TDD generated by SIGALI in order to plug other verification tools, e.g. such as ALDEBARAN [Fer84] or FCTOOLS [BRRD96]. So the result can be submitted to the tool sets for further analysis, graphical depiction, strong bisimulation, quotient computing, etc. The plug-in is achieved with the package OPEN/CAESAR.
Obviously, the size of the generated transition systems limits the extensional methods. For instance, *the transformer station on the power network* which is widely used by the French national power network is represented by a transition system with 12 state variables and 22 event variables; that is to say, this transformer station can be represented by an automaton of 2^{12} possible states and 3^{22} arrows.

The intensional methods for bisimulation checking, as proposed in this paper, perfectly fits the spirit of the tool SIGALI: the "polynomial language" provides the programmer with an intermediate language to describe algorithms over sets, in an intensional way, without bothering with the underlying implementation.
We have then improved the ILTS models semantics by implementing the algorithm of Section 2.4 for the bisimulation decision.

4 Conclusion

In this paper, we have presented *Intensional Labeled Transition Systems* intermediate models for discrete event systems. We have studied operations of parallel composition and event hiding, as well as an equivalence criterion based on strong bisimulation semantics.

The aim of this work is to rely on intensional descriptions of the systems for symbolic verification purposes, such as equivalence checking. The intensional approach we proposed has the main advantage to remain at an interesting level of abstraction in which algorithms can entirely be expressed, whereas classic symbolic approaches often suffer from a lack of algorithmic language.

Moreover, the intensional formalism is completely compatible with the symbolic technics, since intensionally described sets can be represented by standard decision diagrams.

Intensional models have already been the subject of previous work [LBBLG91], under the name of *polynomial dynamical systems*. They were the base of the temporal logics verification tool SIGALI. The results of this paper led us to enrich the scope of the verification tool SIGALI by implementing equivalence checking, such as strong bisimulation (trace equivalence, etc. are under development) on the basis of the intensional philosophy. This application is of high interest since SIGNAL is used in a lot of areas (controller synthesis [LBMR96], robotics [RMC97],...) where models equivalence checking, and on coming models reduction functionality, is crucial.

We aim now to focus on intensional approaches in its generality, in the sense that not only polynomials for finite states systems, but also other formalisms on possibly infinite systems can be investigated for the representation of sets, still remaining decidable for e.g. equivalence checking.

Acknowledgements

We are grateful to Michel Le Borgne for stimulating discussions and ideas, as well as for helping us in implemention achievement. We also would like to thank Herve Marchand for his thorough criticism of the draft.

References

[ALGMR95] T.P. Amagbegnon, P. Le Guernic, H. Marchand, and E. Rutten. Signal : The specification of a generic, verified production cell controller. *Formal Development of Reactive Systems – Case Study, Production Cell, Lecture Notes in Computer Science 891, chapitre VII*, pages 115–129, January 1995.

[BC84] G. Berry and L. Cosserat. The ESTEREL synchronous programming language and its mathematical semantics. In *Seminar on Concurrency, Pittsburgh, LNCS 197*, pages 389–448. Springer-Verlag, July 1984.

[BLJ91] A. Benveniste, P. Le Guernic, and C. Jacquemot. Synchronous programming with events and relations: the SIGNAL language and its semantics. *Science of Computer Programming*, 16:103–149, 1991.

[BRRD96] A. Bouali, A. Ressouche, V. Roy, and R. De Simone. The fc2tools set. In *Proc. of the 5th Int. Conf. AMAST'96, Munich, Germany, LNCS 1101*, July 1996.

[Bry89] R. E. Bryant. Symbolic boolean manipulation with ordered binary decision diagrams. *CM computing Surweys*, pages 293–318, September 1989.

[DLB97] B. Dutertre and M. Le Borgne. Sigali: un système de calcul formel pour la vérification de programmes signal. Technical report, Institut de Recherche en Informatique et Systèmes Aléatoires (IRISA), July 1997.

[DR94] R. De Simone and A. Ressouche. Compositional semantics of esterel and verification by compositional reductions. *Proc. CAV'94, LNCS 818*, 1994.

[Dut92] B. Dutertre. *Spécification et preuve de systèmes dynamiques.* PhD thesis, Université de Rennes I, September 1992.

[Fer84] J.-Cl. Fernandez. *ALDEBARAN: un Système de Vérification par Réduction de Processus Communicants.* Thèse de Doctorat, Univ. Joseph Fourier-Grenoble I, France, July 1984.

[HLR93] N. Halbwachs, F. Lagnier, and P. Raymond. Synchronous observers and the verification of reactive systems. In *Proc. of the Third Int. Conf. on Algebraic Methodology and Software Technology AMAST'93, Twente, Springer Verlag*, June 1993.

[KP98] O. Kushnarenko and S. Pinchinat. Intensional approaches for symbolic methods. Technical report, Institut National de Recherche en Informatique et en Automatique (INRIA), 1998. To appear.

[LBBLG91] M. Le Borgne, A. Benveniste, and P. Le Guernic. Polynomial dynamical systems over finite fields. In G. Jacob and F. Lamnabhi-Lagarrigue, editors, *Algebraic Computing in control*, volume 165 of *Lecture Notes in Control and Information Sciences*, pages 212–222, March 1991.

[LBMR96] M. Le Borgne, H. Marchand, and E. Rutten. Formal verification of SIGNAL programs: Application to a power tranformar station controller. In *Proc. of the 5th Int. Conf. AMAST'96, Munich, Germany, LNCS 1101*, pages 270–285, July 1996.

[Mil89] R. Milner. A complete axiomatisation for observational congruence of finite-state behaviours. *Information and Computation*, 81(2):227–247, 1989.

[Par81] D. Park. Concurrency and automata on infinite sequences. In *Proc. 5th GI Conf. on Th. Comp. Sci., LNCS 104*, pages 167–183. Springer-Verlag, March 1981.

[Pla88] J. A. Plaice. *Sémantique et Compilation de LUSTRE, un Langage Déclaratif Asynchrone.* Thèse de Doctorat, I.N.P. de Grenoble, France, May 1988.

[RMC97] E. Rutten, E. Marchand, and F. Chaumette. An experiment with reactive data-flow tasking in active robot vision. *Software – Practice & Experience*, 27(5):599–621, May 1997.

Efficient State Space Search for Time Petri Nets*

Johan Lilius
Åbo Akademi University
Turku Centre for Computer Science and
Department of Computer Science
FIN-20520 Turku
Email: Johan.Lilius@abo.fi

1 Introduction

The design of concurrent and reactive systems is difficult. Therefore several formal approaches have been developed to the help the engineer in the automatic verification of his designs. Many such approaches are based on the use of state-spaces. In these approaches one models the system in some mathematically founded language and the mechanically calculates all the possible states of the system. Such methods however suffer from the *state-space explosion problem*, an instance of "combinatorial explosion". To alleviate this several techniques have been developed, among them reduction techniques that exploit the independence of events (cf. [8] for a good overview). The intuition behind these techniques is that if two events can be fired in any order such that we always end up in the same state then they are independent. If these independent events do not affect the property we are interested in, then it does not matter in which order we execute them. The reduction is obtained by throwing one of these interleavings away, ie. firing a subset of the enabled transitions, called a *persistent set*. Methods based on the independence of events are often called *partial-order reduction methods*.

Partial-order reduction methods have been successfully applied to untimed systems, but for real-time systems less progress has been made. The main problem seems to be the global nature of time, that makes all clocks in the system dependent on each other. In particular the extension of partial-order methods to time Petri nets is hampered by two difficulties: (1) The standard semantics for time Petri nets implicitly stores the firing order of transitions in the timing constraints. This means that the state space of the time Petri net will form a tree. (2) The firing time of *synchronisation transitions*, ie. transitions with more than one immediate predecessor requires the calculation of the firing times of these predecessors. For this we need absolute constraints which lead to an infinite state space.

*A longer version of this paper that includes proofs is available as [10].

In this work we show that as long as we are only interested in reachability of markings, the first problem can essentially be ignored. For the second problem we show that is possible to derive a finite state space from the state space with absolute constraints.

The state-space is typically traversed with a standard depth-first traversal algorithm, that includes a list of states that have already been seen during the traversal. The test is needed so that we can guarantee termination. It is in this test that the state-space explosion causes its problem. To find out whether we have already seen a state we have to search the set of all previously seen states. In a timed system this set will be much larger, because the systems typically exhibit the same untimed state several times with different timing constraints. As a consequence also the same persistent set will be recalculated several times during the analysis. To alleviate these two problems we propose a novel solution: We use the branching prefix [12] to calculate a set of *looping points*, ie. transitions after which the untimed system will return to a state in which it has previously been. In this way we only need to search the set of looping points. We also show how to extract persistent sets from the branching prefix. In this way the persistent sets are calculated once at the beginning of the analysis.

The last contribution is actually of independent interest: it establishes a connection between the partial-order approaches based on explicit state representation [8] and partial-order approaches based on the implicit branching-prefix representation of states [7].

2 Time Petri Nets

Time Petri nets are a simple yet powerful formalism for modeling concurrent systems with time constraints. The following section recalls the basic definitions of time Petri nets, describes a method for enumerating the reachable states, and gives a definition for independence of transitions in a time Petri net.

2.1 Definition A *time Petri net* is a five-tuple $TPN = (P, T, F, SI, m_0)$, where P is the set of *places*, T is the set of *transitions*, $P \cap T = \emptyset$, $F \subseteq (P \times T) \cup (T \times P)$ is the *flow relation*, $SI : T \longrightarrow \mathbb{N} \cup \{\infty\} \times \mathbb{N} \cup \{\infty\}$ is a function called *static interval*, and $m_0 \subseteq P$ is the *initial marking* of the time Petri net. The tuple (P, T, F, m_0) is the *underlying net*. The boundaries of the static interval associated with a transition t are called *earliest firing time* t^ϵ and *latest firing time* t^λ respectively. The *preset* of $x \in P \cup T$ is ${}^\bullet x = \{y \mid yFx\}$ and *postset* is $x^\bullet = \{y \mid xFy\}$. ■ 2.1

2.2 Definition A *state* of a time Petri net $TPN = (P, T, F, SI, m_0)$ is a pair $S = (m, c)$, where m is a marking of TPN, and $c : T \longrightarrow \mathbb{R}$ is called the *clock function*. The *initial state* of TPN is $s_0 = (m_0, c_0)$, where $c_0(t) = 0$.A transition t of a time Petri net is *enabled* at marking m iff ${}^\bullet t \subseteq m$. The set of all enabled transitions at marking m is denoted by $en(m)$. A transition t may *fire* from state $s = (m, c)$ after delay $\delta \in \mathbb{R}$ denoted $fireable(s, (\delta, t))$ iff $t \in en(m)$, $(m \setminus {}^\bullet t) \cap t^\bullet = \emptyset$, $t^\epsilon \leq c(t) + \delta$, and $\forall t' \in en(m) : c(t') + \delta \leq t'^\lambda$. The set of all transitions that may fire from state s is denoted by $fireable(s)$. A transition t *fires after time* δ from state $s = (m, c)$ giving a new state, $s' = (m', c')$, where: $m' = \setminus {}^\bullet t \cup t^\bullet$, and $c'(t) = 0$ if $t \in en(m') - en(m)$, or $c'(t) = c(t) + \delta$ else. This is denoted by

$s' = \mathsf{fire}(s, (t, \delta))$. ■ 2.2

The behavior of a time Petri net is described in terms of a firing schedule.

2.3 Definition A *firing schedule* of a time Petri net is a finite or infinite sequence of pairs of transitions and time values $\sigma = \sigma_1, \sigma_2, \sigma_3, \ldots$ with $\sigma_i = (\delta_i, t_i)$, where t_i are transitions and $\delta_i \in \mathbb{R}$ are their *firing delays*. The firing schedule σ is *fireable* from the initial state s_0 if there exist states $s_1, s_2, s_3, \ldots$ such that: $s_i = \mathsf{fire}(s_{i-1}, \sigma_i)$ $i > 0$. Given a finite firing schedule σ define $\mathsf{time}_i(\sigma) = \Sigma_{k=0}^{i}\delta_k$. We shall require that $\sigma : \Sigma_{k=0}^{\infty}\delta_k \to \infty$. ■ 2.3

From the above discussion it is clear that since we consider time to be a continuous quantity the set of states of a time Petri net is infinite. Below we will show that this infinite state space can be partitioned into equivalence classes that will consist of markings and sets of inequations that constrain the possible occurrence times of the enabled transitions. The idea of constructing equivalence classes of markings was originally introduced in [4] and later refined in [3]. An alternative approach has been proposed in the context of a model-checking algorithm in [14]. Here we shall refine this approach so that it becomes possible to apply partial order reduction techniques that have been developed for untimed systems, directly to timed systems.

Our idea is based on the following observation from [1]: The possible occurrence time of of a transition is fully determined by the occurrence times of its predecessors. Transitions that are causally independent do not affect the occurrence times of each other.

2.4 Definition A *state class* is a pair (m, I), where $m \subseteq P$, and I is a set of constraints over $T \cup \{\underline{0}\}$. A state class describes the constraints on the possible firing times of the enabled transitions in a marking. The constraints in I are of the form $x - y \leq c$ and we shall require that the set is transitively closed. A set of constraints I is *consistent* iff it has a solution, otherwise it is *inconsistent*. ■ 2.4

To be able to refer to the absolute occurrence times of transitions we introduced an auxiliary transition $\underline{0}$. $\underline{0}$ occurs before any other transition in the net and thus acts as a kind of origo. Constraints $x - \underline{0} \leq c$ express absolute timing constraints from this point.

2.5 Definition The initial state class is given by $I_0 = \{t - \underline{0} \geq t^e \mid t \in en(m_0)\} \cup \{t - \underline{0} \leq t^\lambda \mid t \in en(m_0)\}$. We denote with the variable t the occurrence time of transition t. A transition t_f is *fireable* in (m, I) iff $t_f \in en(m)$, and we can fire it earlier than the other enabled transitions, ie. $I \cup \{t_f \leq t \mid t \in en(m)\}$ is a consistent set of constraints, this is written $t_f \in \mathsf{fireable}(m, I)$.

Given a state class (m_i, I_i) we can fire t_f if $t_f \in \mathsf{fireable}(m_i, I_i)$. The successor state class (m_{i+1}, I_{i+1}) is given by:

$$\begin{aligned} m_{i+1} &= (m_i - {}^\bullet t_f) \cup t_f{}^\bullet \\ I_i' &= I_i \cup \{\mathsf{lower}({}^{\bullet\bullet}t, I_i) + t^e \leq t - \underline{0} \leq \mathsf{upper}({}^{\bullet\bullet}t, I_i) + t^\lambda \mid t \in en(m_{i+1}) - en(m_i)\} \\ I_{i+1} &= \mathsf{delete}(I_i', T(I_i') - (en(m_i) - en(m_{i+1})) - {}^{\bullet\bullet}t_f), \end{aligned}$$

where $\mathsf{lower}(T, I) = \max\{c \mid t - \underline{0} \geq c \in I, t \in T\}$ and $\mathsf{upper}(T, I) = \max\{c \mid t - \underline{0} \leq c \in I, t \in T\}$.

The function $\mathsf{delete}(I, V)$ does two things: It first deletes all variables in V from I, and then it calculates the transitive closure of the new set of constraints. ■ 2.5

As mentioned above, the firing rule will lead to an infinite state space, since the bounds on the constraints of enabled transitions will grow without limit as the system evolves (new instances of the transitions will occur later in time). It is however the case, that after a certain limit the system will enter a state from which state on the behavior will be bisimilar, specifically, the relative constraints will be the same. Thus we can define a bisimulation on state classes as follows:

2.6 Definition Two state classes (m, I), (m', I') are bisimilar, denote by $(m, I) \simeq (m', I')$ iff$m = m' \wedge \mathsf{delete}(I, \underline{0}) = \mathsf{delete}(I', \underline{0})$. ■ 2.6

2.7 Theorem $\simeq$ is a bisimulation with finite index. ■ 2.7

To show the soundness and completeness of our state space we have to show that to each schedule we can find a corresponding path, and vice-versa.

2.8 Theorem Given a schedule $\sigma = (\delta_1, t_1), (\delta_2, t_2), \ldots$ with states $s_i = (\mu_i, \mathsf{clock}_i)$ we can construct a path $\pi = (m_0, I_0) \xrightarrow{t_1} (m_1, I_1) \xrightarrow{t_2} \ldots$ such that $\mu_i = m_i \ \forall i \geq 0$, and given a path $\pi = (m_0, I_0) \xrightarrow{t_1} (m_1, I_1) \xrightarrow{t_2} \ldots$ we can construct a schedule $\sigma = (\delta_1, t_1), (\delta_2, t_2), \ldots$ with states $s_i = (\mu_i, \mathsf{clock}_i)$ such that $\mu_i = m_i \ \forall i \geq 0$. ■ 2.8

This concludes our argument that the firing rule of Def. 2.5 correctly preserves the semantics of a time Petri net.

The definition of independence of events in a systems is a behavioral notion that captures the following intuition: Given independent events, it does not matter in which order they are executed, the end result will be the same. In the untimed case independence of transitions can be formalized in the following definition adapted from [8].

2.9 Definition Two transitions t_1 and t_2 are independent, iff for all states s of the state space: (1) if t_1 is enabled in s and $s \xrightarrow{t_1} s'$, then t_2 is enabled in s iff t_2 is also enabled in s'; and (2) if t_1 and t_2 are enabled in s, then there is a unique state s' such that $s \overset{t_1;t_2}{\Rightarrow} s'$, and $s \overset{t_2;t_1}{\Rightarrow} s'$ ■ 2.9

To lift this notion to the timed case we need to take into account the constraints. The way the state classes were defined above allows us to state independence for time Petri nets in terms of the independence in the underlying net.

2.10 Theorem Let t_1, t_2 be independent in the underlying net. Then given paths $(m, I) \xrightarrow{t_1} (m_1, I_1) \xrightarrow{t_2} (m_2, I_2)$ and $(m, I) \xrightarrow{t_2} (m_1', I_1') \xrightarrow{t_1} (m_2', I_2')$ we have that $m_2' = m_2$ and $I_2 = I_2'$. ■ 2.10

The importance of the theorem lies in the fact that it allows us to use the algorithms for the calculation of independent sets of transitions developed for untimed systems (stubborn sets, ample sets, etc.) by simply replacing the enabling condition with the condition for fireability. We will now turn to the question of finding independent sets of transitions.

3 Partial-order reduction

In this section we will present a version of the state space traversal algorithm for time Petri nets that uses the branching prefix to guide its search.

Below we shall first define the branching prefix, then explain how the basic depth first

search algorithm is adapted to run on the branching prefix. Then we will augment it to work on a time Petri net, and finally discuss how to extract persistent sets from the branching prefix.

In a net (S, T, F) elements x_1, x_2 are in *conflict* $(x_1 \# x_2)$ iff $\exists t_1, t_2 : {}^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset$, $(t_1, x_1) \in F^*$, and $(t_2, x_2) \in F^*$.

3.1 Definition An *occurrence net* $ON = (B, E, G)$ is a finitary, acyclic net, where $\forall b \in B : |{}^\bullet b| \leq 1$, and $\forall e \in E : \neg e \# e$. Places B of an occurrence net are called *conditions* and transitions E are called *events*. Preplaces are called *preconditions* and postplaces *postconditions*. We write $\mathsf{Min}(ON)$ for the $\leq$-minimal elements of ON and call these *initial elements*. ■ 3.1

3.2 Definition Let $PN = (P, T, F, m_0)$ be a Petri net. A *branching process* of PN is a pair $\beta = (ON, p)$, where ON is an occurrence net and p is a homomorphism from ON to PN such that $\forall e_1, e_2 \in E : {}^\bullet e_1 = {}^\bullet e_2 \wedge p(e_1) = p(e_2) \implies e_1 = e_1$. A *configuration* C of a branching process is a set of events, s.th: (1) $e \in C \implies \forall e' \leq e : e' \in C$, and (2) $e, e' \in C \implies \neg e \# e'$. Each branching process β has a set of *maximal configurations* $\mathsf{Max}(\beta)$. Given an event e its *local configuration* $[e]$ is the set $\{e' \mid e' \leq e$. For a configuration C, we define a function $\mathsf{Cut}(C) = (C^\bullet \cup \mathsf{Min}(N)) \setminus {}^\bullet C$. The intuition of a cut is that it represents a state of the system. We denote $p(\mathsf{Cut}(C))$ by $\mathsf{Mark}(C)$. Given a branching process β we define a function $\uparrow(\beta, \mathsf{cut})$ as $\uparrow(\beta, \mathsf{cut}) = \beta - \{x \in \beta \mid \exists y \in \mathsf{cut} : x < y\}$. ■ 3.2

It is clear that the representation of the behavior of a system through the maximal branching process may be infinite. However as discovered by McMillan [11], it is possible to find a finite representation, the *branching prefix*, of the branching process of the system that contains all reachable cuts (ie. markings) of the system. This representation was later improved by Esparza, Römer and Vogler [6].

3.3 Definition Let $\prec$ be defined by $C_1 \prec C_2 \Leftrightarrow |C_1| < |C_2|$. An event e is a *cutoff event* of the branching process β iff β contains a local configuration $[e']$ such that: $p(\mathsf{Cut}([e])) = p(\mathsf{Cut}([e']))$, and $[e'] \prec [e]$. The event e' is the *shift-back* event of e and is denoted by e°.The set of cutoff events of a branching process β is denoted by $\mathsf{Cutoff}(\beta)$. We call the marking $p(\mathsf{Cut}([e]))$ of a cutoff event a *cutoff marking*. ■ 3.3

3.4 Definition Given a net N and its maximal branching process β_m, the finite branching prefix β_f is defined as: $\beta_f = \beta_m - \uparrow(\beta_m, \mathsf{Cutoff}(\beta_m))$ ■ 3.4

The branching prefix is an acyclic representation of the causal behavior of the net. The states are represented as cuts, while the cyclic behavior is implicitly represented by the cutoff events. Thus we need to modify the standard depth-first search algorithm to to take into account this extra structure, as shown in figure 1.

The set of enabled events at a cut in the branching prefix is easily obtained by looking at the presets of the cut. The actual management of cuts and cutoff events is encapsulated into the FIRE function. There are two cases depending on the type of the event e. If e is not a cutoff event then $\textsc{fire}(e, \mathsf{cut}) = \mathsf{cut} - {}^\bullet e \cup e^\bullet$. If e is a cutoff event then things are a bit more complicated, because the new cut is not obtained from the postset of the shift-back event e°, but from the cut $\mathsf{Cut}([e^\circ])$. So we need to replace all conditions b in $\mathsf{cut} - {}^\bullet e$ that are instances of places that have instances in $\mathsf{Cut}([e^\circ])$. That is $\textsc{fire}(e, \mathsf{cut}) = \mathsf{cut} - {}^\bullet e - \{b \in \mathsf{cut} - {}^\bullet e \mid \exists b' \in \mathsf{Cut}([e^\circ]) : p(b) = p(b')\} \cup \mathsf{Cut}([e^\circ])$.

```
SEARCH(bp : BranchingProcess, M : marking)
 1  S : Stack
 2  S.PUSH(MIN(bp))
 3  while S ≠ ∅
 4  do
 5      cut ← S.POP();
 6      for each e ∈ ENABLED(bp, cut)
 7      do
 8          cut′ ← FIRE(e, cut)
 9          if MARK(cut) = M
10            then exit
11          fi
12          if ¬SEEN(e, MARK(cut))
13            then S.PUSH(cut′)
14          fi
15      endf
16  endw
```

Figure 1: Searching the branching prefix.

```
TPNSEARCH(N : TimePetrinet, m : Marking)
 1  S : Stack
 2  bp ← BranchingProcess(N)
 3  S.PUSH(< MIN(bp), I_0 >)
 4  while S ≠ ∅
 5  do
 6      < cut, I >← S.POP();
 7      for each e ∈ FIREABLE(bp, cut, I)
 8      do
 9          < cut′, I′ >← FIRE(e, cut, I)
10          if MARK(cut′) = m
11            then exit
12          fi
13          if ¬SEEN(e, MARK(cut), I′)
14            then S.PUSH(< cut′, I′ >)
15          fi
16      endf
17  endw
```

Figure 2: The modified state space search algorithm

The detection of loops in the search in done in the SEEN function. The idea is to exploit the fact that the occurrence of a cutoff event signals that the system is about to loop. Thus we store only markings that occur after cutoff events, and the SEEN function only needs to check a marking in a list of markings indexed by cutoff events, ie. $\text{SEEN}(e, \text{MARK}(cut)) = cutoff(e) \wedge cut \in Seen(e)$.

Naturally the search procedure as defined above does not make much sense for an untimed net, because it will traverse the state-space once and stop. However the state space of a time Petri net a marking is bound to occur several times with a different set of timing constraints, so the fact that we only compare state classes after cutoff events is bound to speed up the search. The search algorithm for time Petri nets is given in figure 2. The function FIREABLE is defined as $\text{FIREABLE}(bp, cut, I) = \{e \in bp \mid \exists t \in fireable(Mark(cut), I) \wedge p(e) = h\}$.

Analogously the FIRE function is defined in terms of the FIRE function above together with the firing rule 2.5 and the SEEN function now needs to compare state classes..

The algorithm in figure 2 improves on performance of the search only by exploiting the cutoff events. To add partial-order reduction to the algorithm the function fireable needs to return a subset of fireable transitions, and this subset should be a persistent set. The intuition behind our approach is that we should search each process of the system. In a process all events that are enabled at the same cut are independent. Thus at a cut we only need to make sure that we fire one transition for each process. Let $\beta : E \to 2^{Max(bp)}$ be a function that attaches to each event the set of processes (maximal configurations) that it participates in. With $\beta^{-1} : Max(bp) \to 2^E$ we denote the function that maps a maximal configuration to its set of events. Then the set of *active* processes at a cut is given by $act(cut) = \bigcup_{e \in en(cut)} \beta(e)$. When we select a set of events to fire, we have to take care that we select one representative for each active process at the cut. For this we introduce a function rep as follows: $\forall c \in act(cut) \quad \exists e \in rep(cut) : e \in \beta^{-1}(c)$ Now recall the definition of a persistent set from [8]:

3.5 Definition A set of transitions T enabled in a state s is *persistent in* s iff, for all nonempty sequences of transitions $s = s_1 \xrightarrow{t_1} s_2 \ldots s_n \xrightarrow{t_n} s_{n+1}$, from s in the state space and including only transitions $t_i \notin T$, t_n is independent in s_n for all transitions in T. ■ 3.5

3.6 Theorem The set $rep(cut)$ is a persistent set. ■ 3.6

Now because of theorem 2.10 we can use the subset of fireable transitions in the set rep as a persistent set in the search algorithm in figure 2. Thus we have, along the lines of [8], that we can use the set rep to search for *local states* of the system. A local state in this case corresponds to the marking of a place.

3.7 Theorem The algorithm in figure 2, where the function $fireable(bp, cut, I)$ has been replaced by the function $rep(bp, cut, I)$ is a complete and correct decision procedure for determining whether a place p is reachable in the time Petri net N. ■ 3.7

However this is not quite enough. What we want is a procedure that finds a marking m, not just a marked place. For this we introduce the notion of *visible* transitions. The set of visible transitions for a marking m are the transitions that lead to or from the marking ie. $vis(m) = {}^\bullet m \cup m^\bullet$. So we need to extend the rep function to return also all fireable visible transitions: $rep(bp, cut, I, m) = fireable(rep(bp, cut, I) \cup {}^\bullet m \cup m^\bullet)$. We then have:

3.8 Theorem The algorithm in figure 2, where the function $fireable(bp, cut, I)$ has been replaced by the function $rep(bp, cut, I, m)$ is a complete and correct decision procedure for determining whether a marking m is reachable in the time Petri net N. ■ 3.8

4 Conclusions

The use of persistent sets is a standard technique in the context of untimed systems (cf. [8]), while for timed systems only a few proposals exists ([2] for timed automata, [14] for time Petri nets). The main difference between our work and [14] is the semantics, which in [14] is finer, because the states contain an implicit ordering of events. The idea of only comparing state classes that arise after firing a cutoff event, is very similar to the idea of detecting *entry nodes* in [9], in that both approaches try to detect the periodicity of the system by statically analysing the untimed system model.

References

1. T. Aura and J. Lilius. Time processes for time Petri nets. In *Proc. of ATPN'97*, vol. 1248 of *LNCS*, pages 136–155. Springer Verlag, 1997.
2. J. Bengtsson, B. Jonsson, J. Lilius, and W. Yi. Partial order reductions for timed system. In *CONCUR'98*, 1998, to appear.
3. B. Berthomieu and M. Diaz. Modelling and verification of time dependent systems using time Petri nets. *IEEE Transactions on Software Engineering*, 17(3):259–273, March 1991.
4. B. Berthomieu and M. Menasche. A state enumeration approach for analyzing time Petri nets. In *Proc. of ATPN'82*, pages 27–56, Italy, 1982.
5. J. Engelfriet. Branching processes of Petri nets. *Acta Informatica*, 28:575–591, 1991.
6. J. Esparza, S. Römer, and W. Volger. An improvement of McMillan's unfolding algorithm. In *Proc. of TACAS'96*, vol. 1055 of *LNCS*, pages 87–106. Springer Verlag, 1996.
7. J. Esparza. Model checking using net unfoldings. *Science of Computer Programming*, 23:151–195, 1994.
8. P. Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems – An Approach to the State-Explosion Problem*, vol. 1032 of *LNCS*. Springer Verlag, 1996.
9. K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. Efficient verification of real-time systems: Compact data structure and state-space reduction. In *Proc. of the 18th IEEE Real-Time Systems Symposium*, pages 14–24, 1997.
10. J. Lilius. Efficient state space search for time petri nets. Research report, TUCS, 1998. Available as: `http://www.abo.fi/~jolilius/papers/essstpn.ps`.
11. K. L. McMillan. A technique of a state space search based on unfolding. *Formal Methods in System Design*, 6(1):45–56, 1995.
12. K. McMillan. Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In *Proc. of CAV'92*, vol. 663 of *LNCS*, pages 164–177. Springer-Verlag, 1992.
13. P. M. Merlin and D. J. Farber. Recoverability of communication protocols — implications of a theoretical study. *IEEE Transactions on Communications*, 24(9):1036–1043, September 1976.
14. T. Yoneda and H. Schlingloff. Efficient verification of parallel real-time systems. *Journal of Formal Methods in System Design*, 11(2):187–215, 1997.

Projectable semantics for Statecharts*

Andrea Maggiolo-Schettini and Simone Tini
Dipartimento di Informatica, Università di Pisa, Corso Italia 40,
56125 Pisa, Italy.
e-mail: {maggiolo,tini}@di.unipi.it

Abstract

It has been proved that it is impossible to combine in one semantics for reactive systems the notions of modularity, causality and synchronous hypothesis. This limits bottom-up development of specifications. In this paper we introduce the notion of projectability, which is weaker than modularity, we define a non global consistent semantics for Statecharts that enforces projectability, causality and synchronous hypothesis, and we prove that no global consistent semantics for Statecharts can enforce these three notions.

1 Introduction

The visual formalism Statecharts has been proposed in [4] for the specification of *reactive systems* [3], i. e. systems that maintain an ongoing interaction with their environment by continuously reacting to external stimuli.

Statecharts extends state-transitions diagrams with the notions of hierarchy, explicit representation of parallelism and broadcast communication of signals.

According to the principle of *synchronous hypothesis* [2], a statechart is supposed to react instantaneously to prompts from its environment. As a consequence, inputs from the environment and outputs of a statechart come instantaneously.

In [6] properties of *causality* and *modularity* for formalisms that enforce the synchronous hypothesis, have been investigated. Causality means that for each event generated by a system at a particular moment, there exists an event generated by its environment that directly or indirectly causes it. Causality ensures that reactive systems are really driven by their environment. Modularity means, firstly, that if two systems are put together

*Research partially supported by Esprit BRA 8130 LOMAPS and by CNR Progetto Strategico "Modelli e Metodi per la Matematica e l'Ingegneria".

to form a new one, they see each other behaviors as sequences of input-output pairs exactly as the environment sees them. No inner details of the execution of a system can be seen by the other. A second aspect is the uniformity of the view every subsystem has, namely that when an event is generated it is broadcast all around, and every subsystem has the same view at any moment. Finally, a reaction of the compound system is a combination of reactions of its subsystems. This means that the behavior of a system is defined once and for all, and one can freely insert this in whatsoever context, being sure that it maintains its behaviors. This is needed to develop bottom-up specifications. Unfortunately, in [6] it is proved that synchronous hypothesis, causality and modularity cannot be combined in one semantics.

In this paper we introduce a notion weaker than that of modularity, the notion of *projectability*. Projectability does not require that the composition of subsystems is defined by abstracting from causality of their internal events, so one may combine synchronous hypothesis, projectability and causality.

Now, two kinds of semantics have been proposed for Statecharts, non global and global consistent ones. We define a new non global consistent semantics for Statecharts and we prove that it enforces projectability, causality and synchronous hypothesis. Then we prove that there is no global consistent semantics enjoying the same property.

2 Statecharts

Statecharts are state-transitions diagrams with a tree-like structuring of states. States may be basic, or-states, and and-states, allowing to represent parallelism. A transition between two states is labeled by a set of positive and negative signals, the trigger, and a set of positive signals, the action of the transition. Here we assume that source and target state of a transition are both immediate substates (in the tree-like structure) of the same or-state, namely transitions cannot cross borders of states.

Formally, a statechart z is a tuple

$$\langle S_z, \rho_z, \phi_z, \delta_z, T_z, in_z, out_z, \Pi_z, \chi_z \rangle$$

where:

1. S_z is the non-empty, finite *set of states*.
2. $\rho_z : S_z \to 2^{S_z}$ is the *hierarchy function*; for $s \in S_z$, $\rho_z^*(s)$ denotes the least $S \subseteq S_z$ such that $s \in S$ and $\rho_z(s') \subseteq S$ for all $s' \in S$, and $\rho_z^+(s)$ denotes $\rho_z^*(s) \Leftrightarrow \{s\}$; ρ_z describes a tree-like structure, namely:
   (a) There exists a unique $s \in S_z$, denoted $root_z$, s. t. $\rho_z^*(s) = S_z$.
   (b) $s \notin \rho_z^+(s)$, for $s \in S_z$.

(c) If $\rho_z^*(s) \cap \rho_z^*(s') \neq \emptyset$, then either $s' \in \rho_z^*(s)$ or $s \in \rho_z^*(s')$, for $s, s' \in S_z$.

A state s is *basic* iff $\rho_z(s) = \emptyset$.

3. $\phi_z : S_z \rightarrow \{OR, AND\}$ is the (partial) *state type function* defined only for all non-basic states. States with type OR are called *or-states*, states with type AND are called *and-states*.

4. $\delta_z : S_z \rightarrow S_z$ is the (partial) *default function* defined only for or-states, so that $s' = \delta_z(s)$ implies that $s' \in \rho_z(s)$. For $s \in S_z$, $\delta_z^*(s)$ denotes the least $S \subseteq S_z$ s. t. $s \in S$, for each $s' \in S$ of type AND $\rho_z(s') \subseteq S$ and for each $s' \in S$ of type OR $\delta_z(s') \in S$.

5. T_z is the finite *set of transitions*.

6. $in_z, out_z : T_z \rightarrow S_z \Leftrightarrow \{root_z\}$ are the *target* and the *source functions*. It is required that for each $t \in T_z$ there exists a state $s \in S_z$ such that $\phi_z(s) = OR$ and $in_z(t), out_z(t) \in \rho_z(s)$.

7. Π_z is the finite set of *signals*. For each $a \in \Pi_z$, $\overline{a}$ denotes the negation of a. For each $Y \subset \Pi_z$, $\overline{Y}$ is the set $\{\overline{a} | a \in Y\}$.

8. $\chi_z : T_z \rightarrow 2^{\Pi_z \cup \overline{\Pi_z}} \times 2^{\Pi_z}$ is the *labeling function*; the first component of $\chi_z(t)$ is denoted by $trigger(t)$ and is the *trigger* of t, the second component of $\chi_z(t)$ is denoted by $action(t)$ and is the *action* of t.

Given states s_1, s_2 of a statechart z, $lca_z(s_1, s_2)$ denotes the lowest common ancestor of s_1 and s_2, i. e. the state s such that $s_1, s_2 \in \rho_z^*(s)$, and for each $s' \neq s$ fulfilling the same requirement, $s \in \rho_z^+(s')$. For a transition t, $lca_z(t)$ denotes the state $lca_z(in_z(t), out_z(t))$.

The limiting assumption that transitions do not cross borders of states seems to be natural if one wants bottom-up development of specifications.

Given a state s, we denote by $trans(s)$ the set of all the transitions t s. t. $lca_z(t)$ is a substate of s.

The semantics of a statechart is given in terms of steps that take the statechart from a configuration to another.

A *configuration* of a statechart is a maximal set of states fulfilling the requirement that if an and-state is in the configuration, then all its substates are in it, and if an or-state is in the configuration, then exactly one of its substates is in it. The *default configuration* is the configuration such that for each or-state in it, its *default-state* (given by the default function) is in the configuration.

At each instant of time the environment prompts the statechart with a set of signals. Signals are assumed to be broadcast.

A transition is *triggered* by a set of signals if all positive signals of its trigger are communicated and no signal appearing negated in the trigger is communicated. A triggered transition may *fire* and broadcast the signals in its action.

The statechart reacts to a prompt from the environment by performing a set of transitions, called a *step*. When a step $\mathcal{T}$ is performed from configuration C, a new configuration $C' = (C \Leftrightarrow \bigcup_{t\in\mathcal{T}} \rho_z^*(out_z(t))) \cup \bigcup_{t\in\mathcal{T}} \delta_z^*(t)$ is entered. In order to have finite reactions, it is required that for each pair of transitions t, t' in a step $\mathcal{T}$, t and t' are *consistent*, i. e. $\phi_z(lca(t), lca(t')) = AND$. As the synchronous hypothesis is assumed, it is mandatory to have only finite reactions.

Now, since the introduction of the formalism, various semantics for Statecharts have been proposed. In [1] most of them are compared and related. As already mentioned, the semantics proposed can be non global consistent (see the semantics in [5]) and global consistent (see the semantics in [11], [9], [10]), depending on the interpretation of negative signals.

In non global consistent semantics negation is interpreted as "not yet". Steps are computed as sequences of sets of transitions (microsteps) $\mathcal{T} = T_1, \ldots, T_k$ such that for each $t, t' \in \mathcal{T}$, t and t' are consistent and all transitions in T_{i+1} are triggered by signals communicated by either the environment or transitions in $T_1, \ldots, T_i$, for $1 \leq i < k$. Now, transition t having $\overline{a}$ in its trigger and transition t' having a in its action can be in a step $\mathcal{T} = T_1, \ldots, T_k$, provided that t is in a microstep T_i and t' is in a microstep T_j, with $i < j$.

In global consistent semantics negation is interpreted as "never". Steps are computed as fixpoints of some equations and in a step there are never a transition t with $\overline{a}$ in its trigger and a transition t' communicating a.

In [1] it is argued that non global consistent semantics allow to distinguish clearly a cause from its effect, and therefore are more intuitive. The idea is that a sequence of microsteps defines a partial order among transitions, and this order reflects causality.

On the contrary, global consistent semantics allow to have a logical view of signals. Signals can be interpreted as boolean variables, and steps can be computed as solutions of sets of boolean equations. Causality is enforced by considering only minimal solutions. This approach needs to reject programs giving rise to equations systems having no solution for some input.

3 Projectability

Consider now the statechart z_1 in Fig. 1 with the non global consistent semantics of [5] explained above. There exists a deterministic choice between transitions t_1 and t_2, and a deterministic choice between t_3 and t_4. Statechart z_1 reacts to σ either by performing step $\{t_2, t_4\}$ if $a \in \sigma$ or by performing step $\{t_1, t_3\}$ otherwise. Step $\{t_1, t_3\}$ can be computed either as the sequence of microsteps $\{t_1\}$, $\{t_3\}$, or as the sequence $\{t_3\}$, $\{t_1\}$,

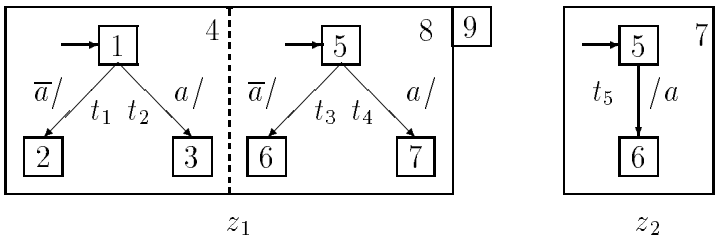


Figure 1:

or as the sequence $\{t_1, t_3\}$. The step $\{t_2, t_4\}$ is computed analogously. There are no other steps and the reachable configurations are only $\{1,4,5,8,9\}$, $\{2,4,6,8,9\}$ and $\{3,4,7,8,9\}$.

Let us consider now the statechart z obtained by composing in parallel z_1 and z_2. If the environment prompts the empty set of signals, then step $\{t_1, t_4, t_5\}$ can be computed as the sequence of microsteps $\{t_1\}$, $\{t_5\}$, $\{t_4\}$. Now, z_1 performs the set of transitions $\{t_1, t_4\}$, which is none of its steps, and reaches the configuration $\{2,4,7,8,9\}$ which is none of its reachable configurations.

For the development of specifications in a bottom-up fashion one reasonably requires that subsystems perform their tasks for which they have been designed, and only these, when inserted in whatsoever context.

Let us consider now the notion of modularity of a reactive system, introduced in [6]. Let $S \overset{\langle I,O\rangle}{\Leftrightarrow\!\!\!\rightarrow} S'$ denote the fact that the reactive system S reacts instantaneously to input I by responding with output O, and by rewriting itself into S'. We denote by $S_1 \parallel S_2$ the parallel composition of S_1 and S_2. A semantics is *modular* iff the following condition holds:

$$(S_1 \overset{\langle I\cup O_2, O_1\rangle}{\Leftrightarrow\!\!\!\rightarrow} S_1' \wedge S_2 \overset{\langle I\cup O_1, O_2\rangle}{\Leftrightarrow\!\!\!\rightarrow} S_2') \Leftrightarrow S_1 \parallel S_2 \overset{\langle I, O_1\cup O_2\rangle}{\Leftrightarrow\!\!\!\rightarrow} S_1' \parallel S_2'. \qquad (1)$$

When S_1 and S_2 are composed in parallel, they see each other as a sequence of pairs $\langle I, O\rangle$, exactly as the environment sees them. The parallel composition of S_1 and S_2 is defined by considering only their input-output interface, i. e. both S_1 and S_2 are viewed as "black boxes", and no inner detail of the execution of one of them is known by the other. Moreover, the output of one system is immediately available as input to the other. This implies the uniformity of the view every subsystem has of what is going on. In [6] it is proved that modularity and causality cannot be combined with synchronous hypothesis. To see this fact, let us assume that $S_1 \overset{\langle\{a\},\{b\}\rangle}{\Leftrightarrow\!\!\!\rightarrow} S_1'$ and $S_2 \overset{\langle\{b\},\{a\}\rangle}{\Leftrightarrow\!\!\!\rightarrow} S_2'$. Modularity implies that $S_1 \parallel S_2 \overset{\langle\emptyset,\{a,b\}\rangle}{\Leftrightarrow\!\!\!\rightarrow} S_1' \parallel S_2'$. Therefore there exists a causal loop between a and b.

The semantics of Esterel [2] and Argos [8] are modular, and programs in which causality loops may occur are rejected.

Another aspect of modularity is that each reaction of the system $S_1 \parallel S_2$ is the union of a reaction of S_1 and a reaction of S_2. The consequence is that the semantics of S_1 (resp. S_2) viewed as a complete system is preserved when it runs in parallel with S_2 (resp. S_1). In this case we are sure that S_1 (resp. S_2) reaches configurations that are reachable also when it runs as a complete system.

The notion of projectability coincides with this aspect of modularity. Formally, a semantics is *projectable* iff the following condition holds:

$$S_1 \parallel S_2 \overset{\langle I, O_1 \cup O_2 \rangle}{\Leftrightarrow\!\!\!\rightarrow} S_1' \parallel S_2' \implies (S_1 \overset{\langle I \cup O_2, O_1 \rangle}{\Leftrightarrow\!\!\!\rightarrow} S_1' \wedge S_2 \overset{\langle I \cup O_1, O_2 \rangle}{\Leftrightarrow\!\!\!\rightarrow} S_2') \tag{2}$$

(which means, obviously, that a modular semantics is projectable but not viceversa).

In the case of Statecharts, we must take care of the hierarchy when defining the notion of projectability.

Definition 1 *A semantics for Statecharts is* projectable *iff given a statechart z and a step $\mathcal{T}$ from configuration C to configuration C', then for each state $s \in C \cap C'$, the set of transitions $\mathcal{T} \cap trans(s)$ is a step of the statechart having s as root-state.*

We give now our non global consistent semantics for Statecharts.

First of all we give our definition of microstep.

Definition 2 *For a statechart z in a configuration C, a sequence of (already) fired sets of transitions $\mathcal{T} = T_1, \ldots, T_k$ and a set of signals $\sigma \supseteq \bigcup_{t \in \mathcal{T}} action(t)$, a set T is a* microstep *iff:*

1. *for each $t \in T$, t is triggered by σ;*
2. *for each $t, t' \in \mathcal{T} \cup T$, t and t' are consistent;*
3. *for each state $s \in C$, it holds that if $trans(s) \cap (\mathcal{T} \cup T) \neq \emptyset$ then there does not exist any transition t such that:*
   (a) *$t \notin \mathcal{T} \cup T$;*
   (b) *t is triggered by the set of signals $\Pi_z \cap (\bigcup_{t \in \mathcal{T} \cap trans(s)} (trigger(t) \cup action(t)) \cup \bigcup_{t \in T \cap trans(s)} trigger(t))$ and t, t' are consistent for each $t' \in \mathcal{T} \cup T$;*
   (c) *$\exists a \in \Pi_z \,|\, (\overline{a} \in trigger(t) \wedge \exists t' \in \mathcal{T} \cap trans(s)$ s. t. $\overline{a} \in trigger(t') \wedge \exists t'' \in (\mathcal{T} \cup T) \Leftrightarrow trans(s)$ s. t. $a \in action(t''))$;*
4. *for each state $s \in C$, it holds that if $t' \in \mathcal{T} \cap trans(s)$, $t \in T \cap trans(s)$, $a \in \Pi_z \cap trigger(t)$, $\overline{a} \in trigger(t')$, then there exists $t'' \in \mathcal{T} \cap trans(s)$ with $a \in action(t'')$.*

Configuration $C' = C \Leftrightarrow \{\rho_z^*(out_z(t)) \,|\, t \in T\} \cup \delta_z^*(in_z(t))$ is reached *from C by means of T.*

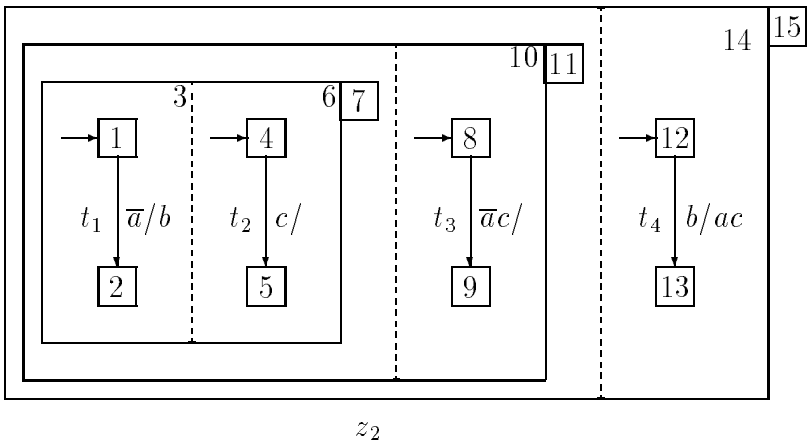


Figure 2:

Condition 3 ensures that given transitions $t, t' \in trans(s)$, both having $\overline{a} \in \overline{\Pi_z}$ in their trigger, $t' \in \mathcal{T} \cup T$, $t \notin \mathcal{T} \cup T$, t triggered by the set of signals $(\bigcup_{t \in \mathcal{T} \cap trans(s)}(trigger(t) \cup action(t)) \cup \bigcup_{t \in T \cap trans(b)} trigger(t)) \cap \Pi_z$, then there does not exist any transition $t'' \notin trans(s)$ with $t'' \in \mathcal{T} \cup T$ and $a \in action(t'')$. The reason is that if the statechart having s as root-state performs t' and transition t is triggered, then either a transition in $trans(s)$ communicates a signal that disallows t or t is performed. Condition 4 ensures that given transitions $t, t' \in trans(s)$ with $t \in T$ and $t' \in \mathcal{T}$ and a signal $a \in \Pi_z$ with $a \in trigger(t)$ and $\overline{a} \in trigger(t')$, then signal a is produced by another transition of s. The reason is that if s, when viewed as a complete statechart, performs t', then it needs to produce a transition having a in its action in order to trigger t.

Definition 3 *Given configurations $C_0, C_1, \ldots, C_n$, a set of signals σ, sets of transitions $T_1, \ldots, T_n$ such that:*

1. *T_{i+1} is a microstep for z in configuration C_i, sets of transitions $T_1, \ldots, T_i$, set of signals $\sigma \cup \{actions(t) | t \in T_1 \cup \ldots \cup T_i\}$, $0 \leq i \leq n \Leftrightarrow 1$;*

2. *C_{i+1} is reached from C_i by means of T_{i+1}, $0 \leq i \leq n \Leftrightarrow 1$;*

3. *there does not exist any microstep $T \neq \emptyset$ for z in configuration C_n, sets of transitions $T_1, \ldots, T_n$, set of signals $\sigma \cup \{actions(t) | t \in T_1 \cup \ldots \cup T_n\}$,*

$\mathcal{T} = T_1, \ldots, T_n$ is a step *for z in configuration C_0, and C_n is the configuration* reached *from C_0 by means of T.*

As an example, let us consider statechart z_2 in Fig. 2 in its initial configuration. If the environment prompts the empty set of signals, the first microstep is $\{t_1\}$, the next is $\{t_4\}$. Now $\{t_1\}, \{t_4\}$ is a step, computed as sequence of microsteps $\{t_1\}, \{t_4\}$.

Transition t_2 is triggered, but all positive signals appearing in the trigger and in the action of t_1 and in the trigger of t_2, trigger t_3 which is not triggered due to the presence of signal a produced by t_4. So $\{t_2\}$ is not a microstep for condition 3 of Def. 3.

Proposition 1 *The semantics of definitions 2 and 3 is projectable.*

Proof. Let us suppose that $\mathcal{T} = T_1, \ldots, T_n$ is a step from configuration C of statechart z, where $s \in C$. Now let us consider the set of signals $\sigma = \bigcup_{t \in \mathcal{T} \cap trans(s)} \{a \in (trigger(t) \cup action(t)) \cap \Pi_z \mid \nexists t' \in \mathcal{T} \cap trans(s). \overline{a} \in trigger(t')\}$. For such environment state s viewed as a complete statechart can execute the sequence of microsteps $trans(s) \cap T_1, \ldots, trans(s) \cap T_n$. Let us consider the set of transitions $T' = T_k \cap trans(s)$ and the sequence of fired sets of transitions $\mathcal{T}' = T_1 \cap trans(s), \ldots, T_{k-1} \cap trans(s)$ for some $1 \leq k \leq n$. The set T' satisfies condition 1 in Def. 2. In fact, if there exists a transition $t' \in T'$ s. t. t' is not triggered by $\sigma \cup \bigcup_{t \in T'} action(t)$, then T_k does not satisfy condition 4 of Def. 2. The fact that T_k satisfies conditions 2, 3, and 4 of Def. 2 implies that T' satisfies the same conditions. Now, assume that the sequence of microsteps $trans(s) \cap T_1, \ldots, trans(s) \cap T_n$ is not maximal. Then there exists a transition $t \in trans(s)$, $t \notin \mathcal{T}$, which is triggered by $\sigma \cup \{a \mid \exists t' \in \mathcal{T} \cap trans(s). a \in action(t)\}$. So there must be $\overline{a} \in trigger(t) \cap \overline{\Pi_z}$ such that $a \in action(t')$ for some $t' \in \mathcal{T} \Leftrightarrow trans(s)$. Now there can be two cases:

1. $\nexists t'' \in \mathcal{T} \cap trans(s)$ with $\overline{a} \in trigger(t'')$. In this case we put $\sigma = \sigma \cup \{a\}$ and reiterate the reasoning.

2. $\exists t'' \in \mathcal{T} \cup trans(s)$ with $\overline{a} \in trigger(t)$. In this case condition 3 of Def. 2 is not satisfied for some microstep in $\mathcal{T}$.

This completes the proof.

Following [12], we could easily give a compositional formalization of the semantics of definitions 2 and 3 by means of Labeled Transition Systems.

Note that in general compositionality does not imply projectability, as it is shown by the compositional semantics in [9], where a reaction of a compound system is obtained by combining "incomplete" reactions of its subsystems.

In global consistent semantics all transitions in a step $\mathcal{T}$ must be triggered by signals communicated by both the environment and transitions of $\mathcal{T}$. As already noticed in [6] global consistency and modularity lead to semantical problems. Assume that we have $S_1 \overset{\langle\{\overline{a}\},\{b\}\rangle}{\Leftrightarrow} S_1'$ and $S_2 \overset{\langle\{b\},\{a\}\rangle}{\Leftrightarrow} S_2'$. No reaction is defined when the compound system $S_1 \parallel S_2$ is prompted with the empty set of signals. In this case it is said that $S_1 \parallel S_2$ has a *non reactive* behavior, in the sense that the system is not able to respond to the environment.

Esterel and Argos reject programs that may have non reactive behaviors.

Figure 3:

The philosophy of Statecharts seems to be contrary to rejecting behaviors at the syntactical level. The original semantics of [11] provides the lack of a reaction in cases like the one of the example above. Two semantics have been proposed that enforce reactivity, namely that assign a reactive behavior to every specification (see [9] and [10]). According to the semantics of [9], the compound system $S_1 \parallel S_2$ as above would react to the empty input by communicating signal b and be rewritten into $S_1' \parallel S_2$. The approach in [10] is that $S_1 \parallel S_2$ reacts to the empty input by performing the empty reaction, i. e. it communicates no signal and it is rewritten into itself.

So, for both the semantics, when one composes two statecharts, the compound step does always exist. However, such semantics may provide that in some environment one component of the system does not make any transition even though for each environment such component considered as a whole system never performs the empty step. For this reason one cannot expect to have projectability.

Proposition 2 *No global consistent semantics can enforce reactivity, causality, projectability and synchronous hypothesis.*

Proof. Let us consider the two statecharts z_1 and z_2 in Fig. 3. If we consider z_1, for each input set of signals either t_1 or t_2 is triggered and therefore performed. Analogously, if we consider z_2, for each input set of signals either t_3 or t_4 is triggered and therefore performed. Now, let us consider the statechart z obtained by composing z_1 and z_2 in parallel. Assume that z performs step $\mathcal{T}$ from its default configuration for the empty input set of signals. Projectability implies that each step $\mathcal{T}$ must satisfy the following condition: $\mathcal{T} \cap \{t_1, t_2\} \neq \emptyset$, $\mathcal{T} \cap \{t_3, t_4\} \neq \emptyset$. Global consistency implies that $\mathcal{T} \neq \{t_1, t_3\}$ and $\mathcal{T} \neq \{t_1, t_4\}$. Causality implies that $\mathcal{T} \neq \{t_2, t_3\}$ and $\mathcal{T} \neq \{t_2, t_4\}$. Therefore no step $\mathcal{T}$ exists.

References

[1] M. von der Beek: *A Comparison of Statecharts Variants.* Lecture Notes in Computer Science 863, pp 128-148, Springer, Berlin, 1994.

[2] G. Berry and G. Gonthier: *The ESTEREL Synchronous Programming Language: Design, Semantics, Implementation.* Science of Computer Programming **19**, pp. 87-152, 1992.

[3] D. Harel and A. Pnueli: *On the Development of Reactive Systems.* In K.R. Apt, editor, Logic and Models of Concurrent Systems, NATO, ASI-13, pp. 477-498, Springer, Berlin, 1985.

[4] D. Harel: *Statecharts: A Visual Formalism for Complex Systems.* Science of Computer Programming **8**, pp. 231-274, 1987.

[5] D. Harel, A. Pnueli, J. P. Schmidt and R. Sherman: *On the formal Semantics of Statecharts.* In Proc. Second IEEE Symp. on Logic in Computer Science, pp. 54-64, IEEE Computer Society Press, New York, 1987.

[6] C. Huizing and R. Gerth: *Semantics of Reactive Systems in Abstract Time.* Lecture Notes in Computer Science 600, Springer, Berlin, pp. 291-314, 1992.

[7] J. J. M. Hooman, S. Ramesh and W. P. de Roever: *A Compositional Axiomatization of Statecharts.* Theoretical Computer Science **101**, pp 289-335, 1992.

[8] F. Maraninchi: *Operational and Compositional Semantics of Synchronous Automaton Composition.* In Proc. of CONCUR 92, Lecture Notes in Computer Science 630, pp. 550-564, Springer, Berlin, 1992.

[9] A. Maggiolo-Schettini, A. Peron and S. Tini: *Equivalence of Statecharts.* In Proc. of CONCUR 96, Lecture Notes in Computer Science 1119, pp. 687-702, Springer, Berlin, 1996.

[10] J. Philipps and P. Scholtz: Compositional Specification of Embedded Systems with Statecharts. In Proc. of TAPSOFT '97. Lecture Notes in Computer Science 1214, pp. 637-651, Springer, Berlin, 1997.

[11] A. Pnueli and M. Shalev: *What is in a Step: on the Semantics of Statecharts.* Lecture Notes in Computer Science 526, pp. 244-264, Springer, Berlin, 1991.

[12] A. C. Uselton and S. A. Smolka: *A Process Algebraic Semantics for Statecharts.* In Proc. of CONCUR 94, Lecture Notes in Computer Science 836, pp. 2-17, Springer, Berlin, 1994.

Strict Lower Bounds for Model Checking BPA

Richard Mayr

Institut für Informatik, Technische Universität München,
Arcisstr. 21, D-80290 München, Germany;
e-mail: mayrri@informatik.tu-muenchen.de
fax: +49 (89) 289-28207/28483
Web: http://www7.informatik.tu-muenchen.de/~mayrri

Abstract

We show strict lower bounds for the complexity of several model checking problems for BPA and branching-time logics. Model checking with Hennessy-Milner Logic *PSPACE*-hard, while model checking with the modal μ-calculus is *EXPTIME*-hard. By combining these results with already established upper bounds it follows that the model checking problems are *PSPACE*-complete and *EXPTIME*-complete, respectively.

1 Introduction

Basic Process Algebra (BPA) processes were defined by Bergstra and Klop in [BK85]. They are transition systems associated with Greibach normal form (GNF) context-free grammars in which only left-most derivations are permitted. BPA-processes are also called *context-free processes*. They are a subclass of pushdown processes, where the finite control of the pushdown automaton has only one state.

It has been known for some time that model checking pushdown processes with the modal μ-calculus is *EXPTIME*-complete [Wal96a, Wal96b]. Furthermore, the problem is even *EXPTIME*-hard for a fixed formula in the alternation-free modal μ-calculus. For the much simpler logic EF, the model checking problem for pushdown processes is *PSPACE*-complete [BEM97]. Again the hardness result even holds for a fixed EF-formula. For CTL the complexity is only known to be between *PSPACE* and *EXPTIME*.

There is an important difference between BPA and pushdown processes in the complexity of model checking. Burkart and Steffen [BS92] showed that for every fixed formula in the alternation-free modal μ-calculus the model checking problem is polynomial in the size

of the BPA-process. Later Walukiewicz [Wal96a, Wal96b] generalized this result to the full modal μ-calculus. The algorithms for BPA were only exponential in the size of the formula. So far there have been no hardness results for model checking BPA, not even for the full modal μ-calculus. On the other hand model checking finite-state systems with the alternation-free modal μ-calculus is linear [SC93, SW91], and model checking finite-state systems with the full modal μ-calculus is in $\mathcal{NP} \cap$ co-$\mathcal{NP}$ [EJS93, SW91, Mad97] (and so might be polynomial as well). It has thus been conjectured that at least some model checking problems for BPA might be polynomial. Here we show that this is not the case. Even for the simple Hennessy-Milner Logic, model checking BPA is *PSPACE*-hard. For the modal μ-calculus model checking is *EXPTIME*-hard. In fact, this hardness result even holds for the alternation-free modal μ-calculus.

The rest of the paper is structured as follows. In Section 2 we define BPA and the logics that are used here. In Section 3 we show the hardness result for Hennessy-Milner Logic, and in Section 4 we show the hardness result for the alternation-free modal μ-calculus. In Section 5 we present the general picture of the complexity of model checking BPA.

2 Preliminaries

We describe BPA-processes by finite sets of rewrite rules of the form $X \xrightarrow{a} \alpha$, where X is a single symbol, $a \in Act$ is an atomic action and α is a sequence of symbols. The rewriting formalism is prefix-rewriting, which means that the rules are only applied at the leftmost position in the term.

The formulae of Hennessy-Milner Logic have the following syntax:

$$\Phi ::= true \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \langle a \rangle \Phi$$

The denotation $[\![\Phi]\!]$ of a formula Φ is a subset of the set of states Ω that is defined inductively as follows:

$$\begin{aligned} [\![true]\!] &:= \Omega \\ [\![\neg\Phi]\!] &:= \Omega - [\![\Phi]\!] \\ [\![\Phi_1 \wedge \Phi_2]\!] &:= [\![\Phi_1]\!] \cap [\![\Phi_2]\!] \\ [\![\langle a \rangle \Phi]\!] &:= \{s \in \Omega \mid \exists s' \in \Omega.\ s \xrightarrow{a} s' \wedge s' \in [\![\Phi]\!]\} \end{aligned}$$

The modal μ-calculus [Koz83] is a fixpoint logic. It is the extension of Hennessy-Milner Logic by variables and fixpoint operators. The semantics of formulae is defined w.r.t. a valuation $\mathcal{V} : Var \mapsto 2^{\Omega}$ that assigns every variable X in the logic a set of states which satisfy it.

$$[\![X]\!]_{\mathcal{V}} := \mathcal{V}(X)$$

The syntax and semantics of the minimal fixpoint operator is defined by

$$[\![\mu X.\Phi]\!]_{\mathcal{V}} := \bigcap \{S \subseteq \Omega \mid [\![\Phi]\!]_{\mathcal{V}[X:=S]} \subseteq S\}$$

where

$$\mathcal{V}[X := S](X') := \begin{cases} \mathcal{V}(X'), & \text{if } X \neq X' \\ S, & \text{if } X = X' \end{cases}$$

In model checking we use only *closed formulae*. These are the formulae where every variable is bound by a fixpoint operator. Also there is the restriction that every variable occurs within the scope of an even number of negations. The property that $s \in [\![\Phi]\!]$ is also denoted by $s \models \Phi$.

3 Hardness of Hennessy-Milner Logic

In this section we show that model checking BPA with Hennessy-Milner Logic is *PSPACE*-hard. We do this by reducing the problem of quantified boolean formulae (QBF) to the model checking problem.

Let $n \in \mathbb{N}$ and $x_1, \ldots, x_n$ be boolean variables. W.r. we assume that n is even. A literal is either a variable or the negation of a variable. A clause is a disjunction of literals. The quantified boolean formula Q is given by

$$Q := \exists x_1 \forall x_2 \exists x_3 \ldots \exists x_{n-1} \forall x_n (Q_1 \wedge \ldots \wedge Q_k)$$

where the Q_i are clauses. The problem is if Q is valid. We reduce this problem to the model checking problem.

The intuition is that first we nondeterministically choose values for the variables and then check if these choices satisfy Q. The existential or universal nature of these choices is handled by the Hennessy-Milner Logic formula. Now we define a BPA with initial symbol Z_0.

$$\begin{array}{lcl} Z_0 & \stackrel{c_1}{\rightarrow} & Z_1.X_1 \\ Z_0 & \stackrel{c_1}{\rightarrow} & Z_1.\bar{X}_1 \\ Z_1 & \stackrel{c_2}{\rightarrow} & Z_2.X_2 \\ Z_1 & \stackrel{c_2}{\rightarrow} & Z_2.\bar{X}_2 \\ & \vdots & \\ Z_{n-1} & \stackrel{c_n}{\rightarrow} & Z_n.X_n \\ Z_{n-1} & \stackrel{c_n}{\rightarrow} & Z_n.\bar{X}_n \\ Z_n & \stackrel{z_n}{\rightarrow} & \epsilon \end{array}$$

Furthermore we add rules $X_i \stackrel{q_j}{\rightarrow} X_i$, if the literal x_i is in the clause Q_j for any $1 \leq i \leq n$ and $1 \leq j \leq k$.

In the same way we add rules $\bar{X}_i \stackrel{q_j}{\rightarrow} \bar{X}_i$, if the literal $\neg x_i$ is in the clause Q_j for any $1 \leq i \leq n$ and $1 \leq j \leq k$.

Now we construct the Hennessy-Milner Logic formula. We use the abbreviation $\langle d \rangle^i$ for $\langle d \rangle \ldots \langle d \rangle$ (i times). The formula is

$$\Phi := \langle c_1 \rangle [c_2] \langle c_3 \rangle [c_4] \ldots \langle c_{n-1} \rangle [c_n] \langle z_n \rangle (\Phi_1 \wedge \ldots \wedge \Phi_k)$$

where

$$\Phi_j := \bigvee_{0 \leq i \leq n-1} \langle d \rangle^i \langle q_j \rangle true$$

It follows that Q is valid iff $Z_0 \models \Phi$. Note that the size of Φ is $\mathcal{O}(n^2 k)$. Thus Φ is not a fixed formula, but grows polynomially with the size of Q. Also the BPA process has size $\mathcal{O}(nk)$. Since QBF is *PSPACE*-complete we get the following lemma.

Lemma 3.1 *Model checking BPA with Hennessy-Milner Logic is PSPACE-hard.*

It was shown in [BEM97] that even for the more general logic EF and pushdown processes, model checking can be done in polynomial space. Thus we get the following theorem.

Theorem 3.2 *Model checking BPA with EF or Hennessy-Milner Logic is PSPACE-complete.*

4 Hardness of the Modal μ-Calculus

Walukiewicz [Wal96a, Wal96b] has shown that model checking pushdown processes with the modal μ-calculus is *EXPTIME*-complete. *EXPTIME*-hardness even holds for a fixed formula in the alternation-free modal μ-calculus. This hardness result does not carry over to BPA. In fact, for every fixed modal μ-calculus formula model checking is polynomial in the size of the BPA-process.

Here we show that model checking BPA with general (non-fixed) formulae in the alternation-free modal μ-calculus is *EXPTIME*-hard. This is shown by a reduction from the acceptance problem for linearly space bounded alternating Turing-machines.

An alternating Turing machine (ATM) is described by a tuple $(Q, \Sigma, \delta, q_0, l)$, where Q are the states of the finite control, Σ the tape symbols, δ the transition relation, q_0 the initial state and l is a function that labels states as existential, universal, accepting or rejecting. The computation of an ATM is defined just like the computation of a normal Turing machine, but the acceptance condition is more complex. Since the machine is nondeterministic, the computation can be represented as a computation tree in which the branches represent different possible computations. The states of the finite control of the ATM are assigned labels by the function l as existential, universal, accepting or rejecting. Now the states in the computation tree are labeled as accepting or rejecting by the following rules:

1. A leaf of the computation tree is labeled accepting (rejecting) if the finite control of the ATM in this state is accepting (rejecting).

2. An internal node where the finite control is labeled universal (existential) is accepting if and only if all (at least one) of its successor nodes are (is) accepting. Otherwise it is rejecting.

3. A node is labeled undefined if the label cannot be determined by the other rules. (This only happens if there are infinite branches.)

Without loss of generality let $|\delta(q,a)| = 2$ for every universal state q and symbol a. We choose an arbitrary order on the two elements of $\delta(q,a)$ and call them the first and second successor configuration of (q,a). An ATM M is called linearly bounded if there is a constant k, such that for every word w in the language of M, M has an accepting computation that uses at most $k \cdot |w|$ space. Let $n := k \cdot |w|$. We only consider linearly bounded ATMs and thus avoid the problem of infinite branches and undefined labels. The acceptance problem for linearly bounded alternating Turing-machines is *EXPTIME*-complete [vL90].

The idea is to guess a sequence of configurations of the ATM and to store this sequence in a BPA-process term. A formula in the alternation-free modal μ-calculus is used to check if this sequence represents an accepting computation of the ATM.

Let $M = (Q, \Sigma, \delta, q_0, l)$ be the ATM, w the input word and $n := k \cdot |w|$ the length of the tape. Let M's head be over the first cell of the tape. We construct in polynomial time a BPA Δ with initial state $\#q_0w\#$ and an alternation-free modal μ-calculus formula Φ s.t. M accepts w iff $\#q_0w\# \models \Phi$, w.r.t. Δ. The rules Δ for the BPA are as follows:

$$
\begin{array}{rcll}
\# & \stackrel{put}{\longrightarrow} & T_1.a & \text{for every } a \in \Sigma \\
T_i & \stackrel{put}{\longrightarrow} & T_{i+1}.a & \text{for every } a \in \Sigma,\ 1 \leq i \leq n-1 \\
T_i & \stackrel{put}{\longrightarrow} & T'_i.q & \text{for every } q \in Q,\ 1 \leq i \leq n \\
T'_i & \stackrel{put}{\longrightarrow} & T'_{i+1}.a & \text{for every } a \in \Sigma,\ 1 \leq i \leq n-1 \\
T'_n & \stackrel{put}{\longrightarrow} & \# & \\
\# & \stackrel{\#}{\longrightarrow} & \epsilon & \\
a & \stackrel{a}{\longrightarrow} & \epsilon & \text{for every } a \in \Sigma \\
q & \stackrel{q}{\longrightarrow} & \epsilon & \text{for every } q \in Q \\
a & \stackrel{drop}{\longrightarrow} & \epsilon & \text{for every } a \in \Sigma \\
q & \stackrel{drop}{\longrightarrow} & \epsilon & \text{for every } q \in Q \\
\# & \stackrel{drop}{\longrightarrow} & \epsilon &
\end{array}
$$

The size of this set of rules is $\mathcal{O}(n^2)$. Now in every state where the symbol # is at the top, the state has the form $\#d_1\#d_2\#\ldots\#d_m\#$ where the d_i are configurations of the ATM. Every d_i has the form $u.q.u'$ where $q \in Q$ is the state, u' is the contents of the tape under the head and to the right of it and u is the contents of the tape to the left of the head. (u can be empty, but u' cannot.) Also we have that $length(u) + length(u') = n$.

Now we define some auxiliary formulae: Let $Q_{acc} \subseteq Q$ be the set of accepting states, $Q_{univ} \subseteq Q$ be the set of universal states and $Q_{ex} \subseteq Q$ be the set of existential states. The

formula *accept* means that the top symbol is # and the state in the uppermost configuration d_1 is accepting.

$$accept := \langle \# \rangle \bigvee_{0 \le i \le n-1} \langle drop \rangle^i \bigvee_{q \in Q_{acc}} \langle q \rangle true$$

The formulae *univ* and *ex* are defined in the same way with Q_{univ} or Q_{ex}, respectively. These formulae have size $\mathcal{O}(n^2)$.

The formula *succ* encodes the property that the state has the form $\#d_1\#d_2\#\ldots\#d_m\#$ for some $m \geq 2$ and that the configuration d_1 is a successor configuration of d_2. The actual construction is cumbersome and depends on the ATM M. However, it is easy to see that it can be done with the help of the following formulae: Let $x \in \Sigma \cup Q$ and $i \in \{0, \ldots, n\}$. $\Psi^1_{x,i}$ means that the i-th symbol in d_1 is x.

$$\Psi^1_{x,i} := \langle \# \rangle \langle drop \rangle^i \langle x \rangle true$$

$\Psi^2_{x,i}$ means that the i-th symbol in d_2 is x.

$$\Psi^2_{x,i} := \langle \# \rangle \langle drop \rangle^{n+2+i} \langle x \rangle true$$

(Note that both d_1 and d_2 have length $n+1$, because the state q counts, too.) In the same way formulae $succ_1$ ($succ_2$) can be constructed that mean that the configuration d_1 is the first successor (second successor) of d_2 if d_2 is a universal configuration. The formulae $\Psi^1_{x,i}$, $\Psi^2_{x,i}$ have size $\mathcal{O}(n)$ and the formulae *succ*, $succ_1$ and $succ_2$ have size $\mathcal{O}(n^2)$.

Now we are ready to construct the formula Φ.

$$\begin{array}{lllll} \mu X. & accept & & & \\ & \vee & & & \\ & univ & \wedge & \langle put \rangle^{n+2}(succ_1 \wedge X) & \\ & & \wedge & \langle put \rangle^{n+2}(succ_2 \wedge X) & \\ & \vee & & & \\ & ex & \wedge & \langle put \rangle^{n+2}(succ \wedge X) & \end{array}$$

Note that Φ is a very simple formula, since it uses only one fixpoint operator. Thus it is a formula in the alternation-free modal μ-calculus. The size of Φ is $\mathcal{O}(n^2)$. We have that $\#q_0w\# \models \Phi$ iff M accepts w.

Lemma 4.1 *Model checking BPA with the alternation-free modal μ-calculus is EXPTIME-hard.*

Containment in *EXPTIME* has been shown in [Wal96a, Wal96b]. Thus we get the following theorem.

Theorem 4.2 *Model checking BPA with the full modal μ-calculus and the alternation-free modal μ-calculus is EXPTIME-complete.*

5 Conclusion

The results on the complexity of model checking BPA can be summarized as follows:

BPA	**general**	**fixed formula**
Hennessy-Milner Logic	*PSPACE*-complete	$\in \mathcal{P}$
EF	*PSPACE*-complete	$\in \mathcal{P}$
alt.-free modal μ-calc.	*EXPTIME*-complete	$\in \mathcal{P}$
modal μ-calc.	*EXPTIME*-complete	$\in \mathcal{P}$

In comparison, model checking pushdown processes is harder in the case of fixed formulae.

Pushdown	**general**	**fixed formula**
Hennessy-Milner Logic	*PSPACE*-complete	$\in \mathcal{P}$
EF	*PSPACE*-complete	*PSPACE*-complete
alt.-free modal μ-calc.	*EXPTIME*-complete	*EXPTIME*-complete
modal μ-calc.	*EXPTIME*-complete	*EXPTIME*-complete

These results solve most complexity questions for branching-time logics, except for CTL. The *EXPTIME*-hardness proof in Section 4 does not carry over to CTL. For both BPA and pushdown processes, model checking with CTL is only known to be between *PSPACE* and *EXPTIME*.

To complete the picture, model checking pushdown processes with LTL and the linear-time μ-calculus is *EXPTIME*-complete, but polynomial for every fixed formula [BEM97]. It has been shown in [May98] that *EXPTIME*-hardness even holds for BPA and LTL.

References

[BEM97] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: application to model checking. In *International Conference on Concurrency Theory (CONCUR'97)*, volume 1243 of *LNCS*. Springer Verlag, 1997.

[BK85] J. A. Bergstra and J.W. Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science (TCS)*, 37:77–121, 1985.

[BS92] O. Burkart and B. Steffen. Model checking for context-free processes. In *Proc. of CONCUR'92*, volume 630 of *LNCS*, pages 123–137, 1992.

[EJS93] E. Emerson, C.S. Jutla, and A. Sistla. On model checking for fragments of μ-calculus. In *Proceedings of CAV'93*, volume 697 of *LNCS*, pages 385–396. Springer Verlag, 1993.

[Koz83] D. Kozen. Results on the propositional μ-calculus. *TCS*, 27:333–354, 1983.

[Mad97] A. Mader. *Verification of Modal Properties Using Boolean Equation Systems.* PhD thesis, TU-München, 1997.

[May98] R. Mayr. *Decidability and Complexity of Model Checking Problems for Infinite-State Systems.* PhD thesis, TU-München, 1998.

[SC93] B. Steffen and R. Cleaveland. A linear–time model–checking algorithm for the alternation–free modal mu–calculus. *International Journal on Formal Methods in System Design*, 1, 1993.

[SW91] C. Stirling and D. Walker. Local model checking in the modal μ-calculus. *Theoretical Computer Science*, 89:161–177, 1991.

[vL90] J. van Leeuwen, editor. *Handbook of Theoretical Computer Science: Volume A, Algorithms and Complexity.* Elsevier, 1990.

[Wal96a] I. Walukiewicz. Pushdown processes: games and model checking. In *International Conference on Computer Aided Verification (CAV'96)*, volume 1102 of *LNCS*. Springer Verlag, 1996.

[Wal96b] I. Walukiewicz. Pushdown processes: games and model checking. Technical Report RS-96-54, BRICS, Aarhus, Denmark, 1996. Longer version of a CAV'96 paper.

Derivation of Characteristic Formulae

Markus Müller-Olm

Department of Computer Science
University of Dortmund
44221 Dortmund, Germany
e-mail: mmo@ls5.informatik.uni-dortmund.de

Abstract

We show how modal mu-calculus formulae characterizing finite-state processes up to strong or weak bisimulation can be derived directly from the well-known greatest fixpoint characterizations of the bisimulation relations. Our derivation simplifies earlier proofs for the strong bisimulation case and, by virtue of derivation, immediately generalizes to various other bisimulation-like relations, in particular weak bisimulation.

1 Introduction

By a classic result of Hennessy and Milner [2, 7] two (image-finite) processes are strongly bisimilar if and only if they satisfy exactly the same formulae of a simple modal logic, now often called *Henessy-Milner-Logic* (HML). In particular, for any two non-bisimilar processes P, Q there is an HML formula ϕ satisfied by P but not by Q. This result shows that HML is sufficiently expressive for distinguishing processes up to strong bisimulation. In another sense, however, the expressiveness of HML is too poor: there is in general no single formula, i.e. no *characteristic formula*, satisfied by just the processes bisimilar to a given process P. Bisimulation classes are thus only characterized by sets of formulae.

Graf and Sifakis [1] show that characteristic formulae can be constructed for finite, i.e. non-cyclic, CCS processes in the *modal mu-calculus*, an extension of HML with fixpoint formulae. This result has been extended to finite-state processes by Steffen and Ingólfsdóttir [9, 10]. While Graf and Sifakis considered strong bisimulation and observational congruence, Steffen and Ingólfsdóttir are concerned with the so-called strong divergence preorder of CCS, a variant of strong bisimulation that takes information about divergence (i.e. internal non-termination) into account. It is not difficult to modify the latter in order to obtain characteristic formulae for strong bisimulation. It is, however, less obvious how to construct characteristic formulae for weak bisimulation-like relations. Actually, [9] proposes to treat weak bisimulation by transforming the processes in such a way that weak bisimilarity of the original processes corresponds to strong bisimulation of the transformed

ones. Then the characteristic formulae for strong bisimulation could be applied on the transformed processes. This approach, however, due to the necessity of transformation does not lead to actual characteristic formulae.

The contribution of this paper is a direct derivation of characteristic formulae from the classic greatest fixpoint characterization of (strong and weak) bisimulation. On the one hand, this provides a more elegant proof of the characterization property. On the other hand it immediately indicates how to construct characteristic formulae for other bisimulation-like process relations, like the various divergence relations discussed in [12], in particular for the weak versions.

We proceed as follows. In the next section we define the modal mu-calculus and labeled transition systems as basic model of processes and introduce equation systems. Section 3 defines the notion of strong bisimulation. In the following section we derive a characteristic equation system of a finite-state process from the fixpoint characterization of strong bisimulation. Section 5 generalizes this to weak bisimulation. In the section thereafter we indicate how to construct actual characteristic formulae from characteristic equation systems. The paper finishes with some concluding remarks.

2 Modal mu-Calculus, Processes, and Equation Systems

The modal mu-calculus [4] is a small, yet expressive process logic. It is defined over a given finite set A of *actions*. We consider in this paper modal mu-calculus formulae in positive normal form which are constructed according to the following grammar:

$$\phi \quad ::= \quad \mathsf{true} \mid \mathsf{false} \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \langle a \rangle \phi \mid [a]\phi \mid X \mid \mu X \,.\, \phi \mid \nu X \,.\, \phi$$

Here, X ranges over an infinite set Var of variables and a over the assumed action set A. The two *fixpoint operators* μX and νX bind the respective variable X and we will apply the usual terminology of free and bound variables in a formula, closed formula etc. Moreover, we write for a finite set M of formulae $\bigwedge M$ and $\bigvee M$ for the conjunction and disjunction of the formulae in M. As usual, we agree that $\bigwedge \emptyset = \mathsf{true}$ and $\bigvee \emptyset = \mathsf{false}$.

Modal mu-calculus formulae are interpreted over processes, which are modeled by labeled transition systems with a designated start state. Formally, a *process* is a structure $P = (S, A, \rightarrow_P, s_0)$, where S is a set of *states*, A is the above (finite) set of *actions*, $\rightarrow_P \subseteq S \times A \times S$ is a *transition relation*, and s_0 is the *initial state*. Throughout this paper we assume that the constituting parts of a process named P are S, A, $\rightarrow_P$, and s_0 and the ones of a process named Q are T, A, $\rightarrow_Q$ and t_0. A process P is called *finite-state* if the underlying state set S is finite.

Suppose given a process P for the remainder of this section. The subset of states that satisfy a formula ϕ, denoted by $M_P(\phi)(\rho)$, is inductively defined in Fig. 1. As usual we refer to *environments*, partial mappings $\rho : \mathsf{Var} \stackrel{\text{part.}}{\rightarrow} 2^S$, which interpret at least the free variables of ϕ by subsets of S, in order to explain the meaning of open formulas. For a set

$$
\begin{array}{rcl}
M_P(\mathsf{true})(\rho) & = & S \\
M_P(\mathsf{false})(\rho) & = & \emptyset \\
M_P(\phi_1 \wedge \phi_2)(\rho) & = & M_P(\phi_1)(\rho) \cap M_P(\phi_2)(\rho) \\
M_P(\phi_1 \vee \phi_2)(\rho) & = & M_P(\phi_1)(\rho) \cup M_P(\phi_2)(\rho) \\
M_P(\langle a \rangle \phi)(\rho) & = & \{s \mid \exists s' : s \xrightarrow{a} s' \ \wedge \ s' \in M_P(\phi)(\rho)\} \\
M_P([a]\phi)(\rho) & = & \{s \mid \forall s' : s \xrightarrow{a} s' \ \Rightarrow \ s' \in M_P(\phi)(\rho)\} \\
M_P(X)(\rho) & = & \rho(X) \\
M_P(\mu X \,.\, \phi)(\rho) & = & \bigcap \{x \subseteq S \mid M_P(\phi)(\rho[X \mapsto x]) \subseteq x\} \\
M_P(\nu X \,.\, \phi)(\rho) & = & \bigcup \{x \subseteq S \mid M_P(\phi)(\rho[X \mapsto x]) \supseteq x\}
\end{array}
$$

Figure 1: Semantics of modal mu-calculus

$x \subseteq S$ and a variable $X \in \mathsf{Var}$ we write $\rho[X \mapsto x]$ for the environment that maps X to x and that is defined on a variable $Y \neq X$ iff ρ is defined on Y and maps Y then to $\rho(Y)$.

Intuitively, true and false hold for all resp. no states and $\wedge$ and $\vee$ are interpreted by conjunction and disjunction. As in HML, $\langle a \rangle \phi$ holds for a state s if there is an a-successor of s which satisfies ϕ, and $[a]\phi$ holds for s if all its a-successors, satisfy ϕ. The interpretation of a variable X is as prescribed by the environment. The formula $\mu X \,.\, \phi$, called a *least fixpoint formula*, is interpreted by the smallest subset x of S that recurs when ϕ is interpreted with the substitution of x for X. Similarly, $\nu X \,.\, \phi$, called *greatest fixpoint formula*, is interpreted by the largest such set. Existence of such sets as well as their characterization used in Fig. 1 follows from the well-known Knaster-Tarski fixpoint theorem [11].

As the meaning of a closed formula ϕ does not depend on the environment, we sometimes write $M_P(\phi)$ for $M_P(\phi)(\rho)$ where ρ is an arbitrary environment. The set of processes *satisfying* a given closed formula ϕ is $P(\phi) = \{Q \mid t_0 \in M_Q(\phi)\}$.

We shall also refer to (closed) *equation systems* of modal mu-calculus formulae,

$$
\begin{array}{rcccl}
E & : & X_1 & = & \phi_1 \\
 & & & \vdots & \\
 & & X_n & = & \phi_n \ ,
\end{array}
$$

where $X_1, \ldots, X_n$ are mutually distinct variables and $\phi_1, \ldots, \phi_n$ are mu-calculus formulae having at most $X_1, \ldots, X_n$ as free variables.

An environment $\rho : \{X_1, \ldots, X_n\} \to 2^S$ is *a solution* of an equation system E, if $\rho(X_i) = M_P(\phi_i)(\rho)$. That solutions always exist, is again a consequence of the Knaster-Tarski fixpoint theorem. For, consider the set of environments that are candidates for solutions, $\mathsf{Env}_P = \{\rho \mid \rho : \{X_1, \ldots, X_n\} \to 2^S\}$. Env_P together with the lifting $\sqsubseteq$ of the inclusion order on 2^S, defined by

$$
\rho \sqsubseteq \rho' \quad \text{iff} \quad \rho(X_i) \subseteq \rho'(X_i) \ \text{ for } i = 1, \ldots, n
$$

forms a complete lattice. Now, we can define the *equation functional* $F_P^E : \mathsf{Env}_P \to \mathsf{Env}_P$ by $F_P^E(\rho)(X_i) = M_P(\phi_i)(\rho)$ for $i = 1, \ldots, n$, the fixpoints of which are just the solutions of E. Certainly, F_P^E is monotonic as $M_P(\phi_i)$ is monotonic such that the Knaster-Tarski fixpoint theorem guarantees existence of solutions. In particular, there is the largest solution νF_P^E of E (w.r.t. $\sqsubseteq$), in which we are particularly interested and which we denote by $M_P(E)$. This definition interprets equation systems on the states of a given process P. We lift this to processes by agreeing that a process satisfies an equation system E, if its initial state is in the largest solution of the first equation. Thus the set of processes satisfying equation system E is $P(E) = \{Q \mid t_0 \in M_Q(E)(X_1)\}$.

3 Strong Bisimulation

As transition systems provide a too fine-grained model of processes, various equivalences have been studied in the literature that identify processes on the basis of their behavior. A classic example is strong bisimulation [8, 7] denoted by $\sim$.

Suppose given two processes P and Q. Bisimulation is first defined as a relation between the state sets S and T and then lifted to the processes themselves. A relation $R \subseteq S \times T$ is called a *(strong) bisimulation* if for all $(s,t) \in R$ the following two conditions hold:

a) $\forall a, s' : s \xrightarrow{a}_P s' \Rightarrow \exists t' : t \xrightarrow{a}_Q t' \wedge (s', t') \in R$, and

b) $\forall a, t' : t \xrightarrow{a}_Q t' \Rightarrow \exists s' : s \xrightarrow{a}_P s' \wedge (s', t') \in R$.

Now, $\sim$ is defined to be the union of all bisimulations R. The processes P and Q are called *bisimilar* if $s_0 \sim t_0$. By abuse of notation we denote this relationship by $P \sim Q$ and view $\sim$ also as a relation between processes.

The relation $\sim \subseteq S \times T$ can also be characterized as the greatest fixpoint $\nu F_\sim$ of the following monotonic functional $F_\sim$ on the complete lattice of relations $R \subseteq S \times T$ ordered by set inclusion:

$$F_\sim(R) \stackrel{\text{def}}{=} \{(s,t) \mid s,t \text{ satisfy the two bisimulation conditions a) and b)}\} \ .$$

For, it is easy to see that a relation R is a bisimulation iff $R \subseteq F_\sim(R)$, i.e. if R is a *post-fixpoint* of $F_\sim$. And, by the Knaster-Tarski fixpoint theorem, $\nu F_\sim$ is just the union of all post-fixpoints of $F_\sim$, i.e. bisimulations, and, therefore, equals $\sim$. This also establishes the well-known fact, that $\sim$ is again a bisimulation, viz. the largest one, as the largest fixpoint of $F_\sim$ clearly is also its largest post-fixpoint.

4 Characteristic Equation Systems

Assume now, that a finite-state process P is given, that $s_1, \ldots, s_n$ are its $|S| = n$ states, and that $s_1 = s_0$ is its initial state. The goal of this paper is to show how a formula characterizing P up to strong bisimulation can be derived from the fixpoint characterization of

bisimulation. While the existence and construction of such formulae is well-known [9, 10], their derivation rather than postulation provides a more elegant proof of the characterization property and shows, moreover, how corresponding formulae for other bisimulation-like equivalences and preorders may be constructed. We illustrate this point by treating also weak bisimulation (see Section 5).

Our derivation of characteristic formulae proceeds via a *characteristic equation system*

$$\begin{array}{rcl} E_\sim & : & X_{s_1} = \phi_{s_1}^\sim \\ & & \vdots \\ & & X_{s_n} = \phi_{s_n}^\sim \end{array}$$

consisting of one equation for each of the states $s_1, \ldots, s_n \in S$. The construction of actual characteristic formulae from the characteristic equation system is deferred to Section 6. The goal is to define the formulae $\phi_s^\sim$ such that the largest solution $M_Q(E_\sim)$ of $E_\sim$ on an arbitrary process Q associates the variables X_s just with the states of Q bisimilar to s, i.e. such that $M_Q(E_\sim)(X_s) = \{t \in T \mid s \sim t\}$.

The construction of $E_\sim$ is based on the observation that Env_Q, the set of candidates for solutions of $E_\sim$, is order-isomorphic to $2^{S\times T}$, the set of relations that are candidates to be bisimulations between S and T. Actually, the mapping $\alpha : \mathsf{Env}_Q \to 2^{S\times T}$ defined by

$$\alpha(\rho) \;=\; \{(s,t) \in S \times T \mid t \in \rho(X_s)\}$$

is an order isomorphism between Env_Q and $2^{S\times T}$, the inverse of which is the mapping $\beta : 2^{S\times T} \to \mathsf{Env}_Q$ defined by $\beta(R)(X_s) = \{t \in T \mid (s,t) \in R\}$.

The idea is now to define $E_\sim$ such that $F_\sim$, the bisimulation functional, and $F_Q^{E_\sim}$, the functional belonging to $E_\sim$, are equal up to the isomorphism induced by (α, β), i.e. such that

$$F_Q^{E_\sim} = \beta \circ F_\sim \circ \alpha \;\; . \tag{1}$$

Then their largest fixpoints are also related by the isomorphism, which yields

$$\begin{array}{ll} & M_Q(E_\sim)(X_s) \\ = & \quad \text{[Definition of } M_Q(E_\sim)\text{]} \\ & (\nu F_Q^{E_\sim})(X_s) \\ = & \quad \text{[Fixpoints of } F_Q^{E_\sim} \text{ and } F_\sim \text{ are related by the isomorphism]} \\ & \beta(\nu F_\sim)(X_s) \\ = & \quad \text{[Definition of } \beta\text{]} \\ & \{t \in T \mid (s,t) \in (\nu F_\sim)\} \\ = & \quad [\sim \text{ equals } \nu F_\sim] \\ & \{t \in T \mid s \sim t\} \;\; , \end{array}$$

as required. By the definition of $F_Q^{E\sim}$, (1) amounts to defining ϕ_s such that

$$t \in M_Q(\phi_s^\sim)(\rho) \quad \text{iff} \quad t \in (\beta \circ F_\sim \circ \alpha)(\rho)(X_s) \ .$$

The strategy for achieving this equivalence is to start a calculation with the right hand side and to stepwise transform it into the direction of a formula:

$t \in (\beta \circ F_\sim \circ \alpha)(\rho)(X_s)$

iff [Definition of β]

$(s,t) \in (F_\sim \circ \alpha)(\rho)$

iff [Definition of $F_\sim$]

$\forall a : \forall s' : s \xrightarrow{a}_P s' \ \Rightarrow \ \exists t' : t \xrightarrow{a}_Q t' \wedge (s',t') \in \alpha(\rho)$, and

$\forall a : \forall t' : t \xrightarrow{a}_Q t' \ \Rightarrow \ \exists s' : s \xrightarrow{a}_P s' \wedge (s',t') \in \alpha(\rho)$

iff [Definition of α]

$\forall a : \forall s' : s \xrightarrow{a}_P s' \ \Rightarrow \ \exists t' : t \xrightarrow{a}_Q t' \wedge t' \in \rho(X_{s'})$, and

$\forall a : \forall t' : t \xrightarrow{a}_Q t' \ \Rightarrow \ \exists s' : s \xrightarrow{a}_P s' \wedge t' \in \rho(X_{s'})$

iff [Definition of $M_Q(X_{s'})$]

$\forall a : \forall s' : s \xrightarrow{a}_P s' \ \Rightarrow \ \exists t' : t \xrightarrow{a}_Q t' \wedge t' \in M_Q(X_{s'})(\rho)$, and

$\forall a : \forall t' : t \xrightarrow{a}_Q t' \ \Rightarrow \ \exists s' : s \xrightarrow{a}_P s' \wedge t' \in M_Q(X_{s'})(\rho)$

iff [Definition $\langle a \rangle$, Definition $\bigvee$]

$\forall a : \forall s' : s \xrightarrow{a}_P s' \ \Rightarrow \ t \in M_Q(\langle a \rangle X_{s'})(\rho)$, and

$\forall a : \forall t' : t \xrightarrow{a}_Q t' \ \Rightarrow \ t' \in M_Q(\bigvee\{X_{s'} \mid s \xrightarrow{a}_P s'\})(\rho)$

iff [Definition $\bigwedge$, Definition $[a]$]

$\forall a : t \in M_Q(\bigwedge\{\langle a \rangle X_{s'} \mid s \xrightarrow{a}_P s'\})(\rho)$, and

$\forall a : t \in M_Q([a]\bigvee\{X_{s'} \mid s \xrightarrow{a}_P s'\})(\rho)$

iff [Definition $\bigwedge$]

$t \in M_Q(\bigwedge\{\bigwedge\{\langle a \rangle X_{s'} \mid s \xrightarrow{a}_P s'\} \mid a \in A\})(\rho)$, and

$t \in M_Q(\bigwedge\{[a]\bigvee\{X_{s'} \mid s \xrightarrow{a}_P s'\} \mid a \in A\})(\rho)$

iff [Definition $\wedge$]

$$t \in M_Q(\bigwedge\{\bigwedge\{\langle a \rangle X_{s'} \mid s \xrightarrow{a}_P s'\} \mid a \in A\} \wedge \bigwedge\{[a]\bigvee\{X_{s'} \mid s \xrightarrow{a}_P s'\} \mid a \in A\})(\rho) \ .$$

Thus, (1) becomes valid if we define $\phi_s^\sim$ by

$$\phi_s^\sim \ \stackrel{\text{def}}{=} \ \bigwedge\{\bigwedge\{\langle a \rangle X_{s'} \mid s \xrightarrow{a}_P s'\} \mid a \in A\} \ \wedge \ \bigwedge\{[a]\bigvee\{X_{s'} \mid s \xrightarrow{a}_P s'\} \mid a \in A\}$$

and this gives us the desired theorem.

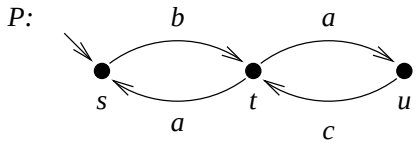


Figure 2: An example process.

Theorem 1 (Char. eq. system on states) $M_Q(E_\sim)(X_s) = \{t \in T \mid s \sim t\}$.

This theorem holds for all processes Q as $E_\sim$ does not depend on Q. In particular, a process Q is bisimilar to P iff its initial state t_0 is contained in $M_Q(E_\sim)(X_{s_1})$ (recall that s_1 is the initial state of P). Thus we have the following corollary.

Corollary 1 (Char. eq. system on processes) $P(E_\sim) = \{Q \mid P \sim Q\}$.

For illustration, we consider the small process pictured in Fig. 2 with state set $S = \{s, t, u\}$ and action alphabet $A = \{a, b, c\}$. After removing conjuncts reading false, its characteristic equation system reads as follows:

$$\begin{array}{rcccl} E_\sim & : & X_s & = & \langle b\rangle X_t \wedge [a]\mathsf{false} \wedge [b]X_t \wedge [c]\mathsf{false} \\ & & X_t & = & \langle a\rangle X_s \wedge \langle a\rangle X_u \wedge [a](X_s \vee X_u) \wedge [b]\mathsf{false} \wedge [c]\mathsf{false} \\ & & X_u & = & \langle c\rangle X_t \wedge [a]\mathsf{false} \wedge [b]\mathsf{false} \wedge [c]X_t \end{array}$$

5 Weak Bisimulation

Strong bisimulation requires that every step of a process is matched by a corresponding step of a bisimilar process. Weak bisimulation [7] denoted by $\approx$ relaxes this requirement for internal computation steps represented by a distinguished action $\tau \in A$, which can be matched by zero of more internal steps. The definition of weak bisimulations relies on a derived transition relation $\stackrel{a}{\Rightarrow}$ that allows arbitrarily many τ-transitions before and after an a-transition. In addition, the relation $\stackrel{\varepsilon}{\Rightarrow}$ is used that represents zero or more τ-transitions:

$$\stackrel{\varepsilon}{\Rightarrow} \stackrel{\text{def}}{=} \stackrel{\tau}{\rightarrow}^* \qquad\qquad \stackrel{a}{\Rightarrow} \stackrel{\text{def}}{=} \stackrel{\varepsilon}{\Rightarrow}; \stackrel{a}{\rightarrow}; \stackrel{\varepsilon}{\Rightarrow}$$

Here, the operator ; denotes relational composition.

Now, a relation $R \subseteq S \times T$ between the state sets of two processes P and Q is called a *weak bisimulation* [7] if for all $(s, t) \in R$ the following two conditions hold:

a) $\forall a, s' : s \stackrel{a}{\rightarrow}_P s' \Rightarrow \exists t' : t \stackrel{\hat{a}}{\Rightarrow}_Q t' \wedge (s', t') \in R$, and

b) $\forall a, t' : t \stackrel{a}{\rightarrow}_Q t' \Rightarrow \exists s' : s \stackrel{\hat{a}}{\Rightarrow}_P s' \wedge (s', t') \in R$.

$\approx$ is defined to be the union of all weak bisimulations R and is the largest weak bisimulation. As for strong bisimulation, P and Q are called bisimilar, $P \approx Q$ for short, if $s_0 \approx t_0$.

Again, we can define a monotonic functional $F_{\approx} : 2^{S\times T} \to 2^{S\times T}$ on relations from the two conditions in the definition of weak bisimulations, the greatest fixpoint of which equals $\approx$. Moreover, an equation system characterizing a process up to weak bisimulation,

$$\begin{array}{rcl} E_{\approx} \;:\; X_{s_1} & = & \phi^{\approx}_{s_1} \\ & \vdots & \\ X_{s_n} & = & \phi^{\approx}_{s_n} \;, \end{array}$$

i.e. that satisfies $M_Q(E_{\approx})(X_s) = \{t \mid s \approx t\}$, can be constructed along the lines of the construction for strong bisimulation. The only difference is the occurrence of the derived transition relations $\stackrel{\hat{a}}{\Rightarrow}$ in the corresponding places. In order to tackle them we rely on 'weak' analogies $\langle\!\langle a \rangle\!\rangle$ of the modality $\langle a \rangle$, which can be introduced as abbreviations:

$$\langle\!\langle \varepsilon \rangle\!\rangle \phi \stackrel{\text{def}}{=} \mu X \,.\, \phi \vee \langle \tau \rangle X \qquad\qquad \langle\!\langle a \rangle\!\rangle \phi \stackrel{\text{def}}{=} \langle\!\langle \varepsilon \rangle\!\rangle \langle a \rangle \langle\!\langle \varepsilon \rangle\!\rangle \phi \;.$$

The following proposition shows that they indeed correspond to $\stackrel{\varepsilon}{\Rightarrow}$ and $\stackrel{a}{\Rightarrow}$.

Proposition 1 (Weak diamond)

- $M_P(\langle\!\langle \varepsilon \rangle\!\rangle \phi)(\rho) \;=\; \{s \mid \exists s' : s \stackrel{\varepsilon}{\Rightarrow}_P s' \;\wedge\; s' \in M_P(\phi)(\rho)\}\,.$
- $M_P(\langle\!\langle a \rangle\!\rangle \phi)(\rho) \;=\; \{s \mid \exists s' : s \stackrel{a}{\Rightarrow}_P s' \;\wedge\; s' \in M_P(\phi)(\rho)\}\,.$

Using these weak modalities it is now straightforward to redo the calculation that lead to an adequate definition of $\phi^{\sim}_s$ also for weak bisimulation, which results in the following definition for $\phi^{\approx}_s$:

$$\begin{array}{rcl} \phi^{\approx}_s & \stackrel{\text{def}}{=} & \bigwedge\{\bigwedge\{\langle\!\langle \hat{a} \rangle\!\rangle X_{s'} \mid s \stackrel{a}{\rightarrow}_P s'\} \mid a \in A\} \;\wedge \\ & & \bigwedge\{[a]\bigvee\{X_{s'} \mid s \stackrel{\hat{a}}{\Rightarrow}_P s'\} \mid a \in A\} \;. \end{array}$$

The derivation shows in particular where to use strong and weak modalities and which set construction have to range over strong and weak successors.

Theorem 2 (Char. eq. system on states) $M_Q(E_{\approx})(X_s) = \{t \in T \mid s \approx t\}$.

Corollary 2 (Char. eq. system on processes) $P(E_{\approx}) = \{Q \mid P \approx Q\}$.

6 Towards Characteristic Formulae

Up to now processes were characterized up to strong or weak bisimulation by an appropriately defined equation system. Actual *characteristic formulae*, i.e. *single* formulae characterizing processes can be constructed by applying simple semantics-preserving transformation rules on equation systems, which are provided in this section. Together these rules allow to reduce an equation system stepwise by ever more equations. These rules are

$$
\begin{array}{lrcl}
F: & X_1 & = & \phi_1 \\
 & & \vdots & \\
 & X_{n-1} & = & \phi_{n-1} \\
 & X_n & = & \nu X_n \,.\, \phi_n
\end{array}
\qquad
\begin{array}{lrcl}
G: & X_1 & = & \phi_1[\phi_n/X_n] \\
 & & \vdots & \\
 & X_{n-1} & = & \phi_{n-1}[\phi_n/X_n] \\
 & X_n & = & \phi_n
\end{array}
\qquad
\begin{array}{lrcl}
H: & X_1 & = & \phi_1 \\
 & & \vdots & \\
 & X_{n-1} & = & \phi_{n-1}
\end{array}
$$

Figure 3: Results of the transformation rules

similar to the ones used by A. Mader in [6] as a means of solving Boolean equation systems (with alternation) by means of Gauss elimination.

In Fig. 3 we show the equation systems resulting from applying each of our three transformation rules on an equation system of the form

$$
\begin{array}{lrcl}
E: & X_1 & = & \phi_1 \\
 & & \vdots & \\
 & X_n & = & \phi_n \;.
\end{array}
$$

For notational convenience, we describe the transformations only w.r.t. the last equation in an equation system.

The first rule, transforming E to F, allows to remove the recursive dependency of the right hand side formula in an equation from the left hand side variable of that same equation. It is not difficult to show that, albeit F might have fewer solutions than E, their greatest solutions coincide on every process Q.

Proposition 2 *$M_Q(E) = M_Q(F)$.*

The second rule, that transforms E to G, allows to replace the variable on the left hand side of an equation by the formula on the right hand side in the other equations. As usual, $\phi[\psi/X]$ denotes the substitution of ψ for the free occurrences of X in ϕ. Being an instance of a substitution of 'equals for equals', E and G have the same solutions, as expected.

Proposition 3 *E and G have the same solutions, in particular, $M_Q(E) = M_Q(G)$.*

Our third and last rule, transforming E to H, allows to remove unnecessary equations from an equation system. It relies on the side condition that the variable X_n does not appear free in $\phi_1, \ldots, \phi_n$. Note that by this side condition H is indeed a closed equation system. Removal of unnecessary equations does not affect the interpretation of the other variables in solutions.

Proposition 4 *Suppose X_n does not appear free in $\phi_1, \ldots, \phi_n$.*

An environment ρ is a solution of H if and only if $\rho[X_n \mapsto M_Q(\phi_n)(\rho)]$ is a solution of E. In particular, $M_Q(E) = M_Q(H)[X_n \mapsto M_Q(\phi_n)(M_Q(H))]$.

Now, applying to an equation system E the first rule followed by the second rule results in an equation system that satisfies the side condition of the third rule. Thus the last equation can be removed; the result is the equation system:

$$\begin{array}{rcccl} \tilde{E} & : & X_1 & = & \phi_1[\nu X_n \,.\, \phi_n / X_n] \\ & & & \vdots & \\ & & X_{n-1} & = & \phi_{n-1}[\nu X_n \,.\, \phi_n / X_n] \ . \end{array}$$

This procedure can be iterated until an equation system with just one equation $X_1 = \psi$ is obtained. A final application of the first rule results in the equation system with just the equation $X_1 = \nu X_1 \,.\, \psi$. The only solution of this equation system on a process Q is the environment ρ defined by $\rho(X_1) = M_Q(\nu X_1 \,.\, \psi)$ as $\nu X_1 \,.\, \psi$ is a closed formula. By the correctness of the transformation rules, $\nu X_1 \,.\, \psi$ is thus a formula, the interpretation of which coincides with the interpretation of X_1 in the greatest solution of the original equation system E. Therefore any set of processes that can be characterized by an equation system can also be characterized by a single formula. Note, however, that the iterated application of the second transformation rule can lead to an exponential blow-up of the size of the formula.

Theorem 3 *For any equation system E there is a formula ϕ such that $P(E) = P(\phi)$.*

This procedure can, in particular, be applied to $E_\sim$ and $E_\approx$ which shows that there are indeed characteristic formulae describing processes up to strong or weak bisimulation.

Theorem 4 (Characteristic formulae) *For any process P there are modal mu-calculus formulae $\psi^\sim$ and $\psi^\approx$ such that $P(\psi^\sim) = \{Q \mid P \sim Q\}$ and $P(\psi^\approx) = \{Q \mid P \approx Q\}$.*

7 Conclusion

We have shown how equation systems and formulae that characterize finite-state processes up to strong or weak bisimulation can be derived directly from the well-known greatest fixpoint characterizations of these relations. The existence of such formulae for strong bisimulation was well-known. By virtue of derivation, however, our simpler and more elegant proof generalizes immediately to weak bisimulation and can, moreover, easily be adapted to various other behavioral equivalences and preorders (like simulation and the preorders studied in [12]).

Do characteristic formulae exist also for some class of infinite-state processes? The answer is no. Any mu-calculus formula ψ representing a certain process P e.g. up to bisimulation has – by the finite model property of the modal mu-calculus [5] – also a finite model Q. Thus P must be bisimilar to Q, i.e. be a finite process up to bisimulation.

What is the use of characteristic equation systems and formulae? On the theoretical side, their existence provides specific expressiveness results for the modal mu-calculus. Combined with the fact that model checking the modal mu-calculus is decidable for certain classes of infinite-state processes, in particular push-down processes, this immediately

implies that strong and weak bisimulation (and various other relations for which characteristic formulae can easily be constructed, e.g. simulation) are decidable between finite-state processes and push-down processes. More far-reaching decidability results of this kind have recently been studied by Jančar, Kučera, and Mayr [3].

On the practical side, characteristic formulae allow to employ model checkers as bisimulation checkers. For this application the exponential blow-up experienced in the transition from characteristic equation systems to characteristic formulae seems to be particularly unfortunate. However, many model checkers are based on equation systems rather than formulae, such that they can be applied directly on characteristic equation systems.

Acknowledgment. I thank Bernhard Steffen for a number of discussions on topics related to this paper and three anonymous referees of MFCS for comments that helped to improve this paper.

References

[1] S. Graf and J. Sifakis. A modal characterization of observational congruence on finite terms of CCS. *Information and Control*, 68:125–145, 1986.

[2] M. C. B. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *JACM*, 32(1):137–161, 1985.

[3] P. Jančar, A. Kučera, and R. Mayr. Deciding bisimulation-like equivalences with finite-state processes. In *Proceedings of ICALP'98*, 1998. To appear in Springer LNCS series.

[4] D. Kozen. Results on the propositional mu-calculus. *TCS*, 27:333–354, 1983.

[5] D. Kozen. A finite model theorem for the propositional μ-calculus. *Studia Logica*, 47:233–241, 1988.

[6] A. Mader. Modal μ-calculus, model checking and Gauss elimination. Volume 1019 of *Lecture Notes in Computer Science*, pages 72–88. Springer-Verlag, 1995.

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

[8] D. M. R. Park. Concurrency and automata on infinite sequences. Volume 154 of *Lecture Notes in Computer Science*, pages 561–572. Springer-Verlag, 1981.

[9] B. Steffen. Characteristic formulae. Volume 372 of *Lecture Notes in Computer Science*, pages 723–732. Springer-Verlag, 1989.

[10] B. Steffen and A. Ingólfsdóttir. Characteristic formulae for processes with divergence. *Information and Computation*, 110(1):149–163, 1994.

[11] A. Tarski. A lattice-theoretical fixpoint theorem and its application. *Pacific Journal of Mathematics*, 5:285–309, 1955.

[12] D. J. Walker. Bisimulations and divergence in CCS. *Information and Computation*, 85(2):202–241, 1990.

Construction of an Abstract State-Space from a Partial-Order Representation of the Concrete One

Ulrich Nitsche
Department of Electronics and Computer Science, University of Southampton
Southampton, SO17 1BJ, United Kingdom, email: un@ecs.soton.ac.uk

Abstract

There are several approaches to reduce a state-space which represents the behaviour of a system that can be classified into two main concepts: *abstraction techniques* and *partial-order methods*. Abstraction reduces the state-space by reducing the diversity of actions. Partial-order methods ignore particular interleavings of concurrent behavioural patterns. Both concepts have certain benefits and drawbacks. We thus present in this paper a first concept to combine partial-order methods and abstraction enabling us to bypass an exhaustive construction of a state-space when constructing an abstract representation of it.

1 Introduction

In practice, the size of automata descriptions of the behaviour of realistic systems limits the application of verification concepts to rather small specifications. To make verification accessible to a much larger group of systems of practical interest, one has to deal with what is known as *state-space explosion*. There are several approaches to reduce a state-space which represents the behaviour of a system. The aim of state-space reduction is to improve the efficiency of verification algorithms. Basically two main concepts for the construction of a reduced state-space exist: *abstraction techniques* and *partial-order methods*.

Abstraction reduces the state-space by ignoring particular actions of the system or reducing the diversity of actions. Partial-order methods ignore particular interleavings of concurrent behavioural patterns. Partial-order methods have the drawback that they are not applicable to some property classes (e.g. liveness properties), abstraction methods require a construction of the complete concrete state-space of a system's behaviour before collapsing it which can be too complex in practice.

We present in this paper a first concept for the use of partial-order methods to define a reduced concrete state-space of a given specification having the same abstract state-space as the complete state-space. Partial-order methods can enable us to bypass an exhaustive construction of the state-space of a specification when constructing a representation of its abstract behaviour.

2 Preliminaries

The *behaviour* of a reactive concurrent system can be represented by a set of infinitely long sequences of actions. Actions are atomic acts the system performs. We consider the set Σ of actions to be finite (from a sufficiently abstract point of view). Thus a behaviour is an ω-language on the set of actions. It represents all sequences of actions that the system can perform in an infinite amount of time. We call each infinite sequence of actions that the system can perform a *computation* of the system.

The sequences of actions that the system can perform in a finite amount of time are the *partial computations* of the system. The *partial behaviour* of a system is the set of finitely long sequences of actions that the system can perform, i.e. it is a language on the set of actions. Since all prefixes of a partial computation of a system are also partial computations of the system, we require that a partial behaviour is a *prefix-closed* language (subsequently let Σ be the finite set of atomic actions that the system may perform):

Definition 2.1 *Let $L \subseteq \Sigma^*$ be a language on Σ. Let $pre(L)$ designate the set of all prefixes of words in L. L is prefix-closed if and only if $pre(L) = L$.*

The behaviour of a system is determined by its partial behaviour continued to infinity. The idea of the infinite "continuation" of a partial behaviour can be defined in terms of formal language theory by the notion of the *Eilenberg-limit* of a language:

Definition 2.2 *Let $L \subseteq \Sigma^*$ be language. The Eilenberg-limit $lim(L)$ of L is defined as $lim(L) = \{x \in \Sigma^\omega \mid \exists^\infty w \in pre(x) : w \in L\}$. Read "$\exists^\infty$..." as "there exist infinitely many different...".*

We consider in this paper only *regular* behaviours, i.e. behaviours that can be represented by a finite automaton [10]. We thus define:

Definition 2.3 *A behaviour of a system is the Eilenberg-limit of a prefix-closed regular language.*

Prefix-closed regular languages and their Eilenberg-limits can be represented by (deterministic) finite automata with only accepting states. The minimal automata representation of a behavoiur is called its *state-space*.

A behaviour satisfies a *linear property* if and only if all its computations satisfy it. Intuitively, a property partitions the set Σ^ω into the set $Y \subseteq \Sigma^\omega$ of the computations

that satisfy the property and the set $N \subseteq \Sigma^\omega$ of computations that do not. To define a property formally, we simply identify the property with the set Y of computations that satisfy it.

Definition 2.4 *A property P over Σ is a subset of Σ^ω; hence it is an ω-language over Σ. We say that a behaviour $lim(L)$, $L \subseteq \Sigma^*$ and $L = pre(L)$, satisfies P (written:"$lim(L) \models P$") if and only if $lim(L) \subseteq P$.*

To introduce an implicit fairness assumption into the satisfaction relation, we define *relative liveness properties* [9, 12, 14] of behaviours as an *approximate satisfaction* relation for properties [13]. To do so, we first have to introduce the notion of a leftquotient [5, 8]:

Definition 2.5 *Let $w \in \Sigma^*$ and let $L \subseteq \Sigma^*$. The leftquotients of L and $lim(L)$ by w are defined as $cont(w, L) = \{v \in \Sigma^* \mid wv \in L\}$ and $cont(w, lim(L)) = \{x \in \Sigma^\omega \mid wx \in lim(L)\}$*

Definition 2.6 *We call the property $\mathcal{P} \subseteq \Sigma^\omega$ a relative liveness property of $lim(L)$ (written: "$lim(L) \models_{RL} \mathcal{P}$") if and only if*

$$\forall w \in pre(lim(L)) : \exists x \in cont(w, lim(L)) : wx \in \mathcal{P}.$$

Note 2.7 *If $lim(L) = \Sigma^\omega$, then the definitions of a relative liveness property is equivalent to the definition of a liveness property in [3].*

Usually, relative liveness and the related concept of machine-closure [1, 2, 4, 9] is used to classify properties with respect to other properties. In contrast we consider relative liveness as a satisfaction relation with an inherent fairness condition that we call *approximate satisfaction* [13]:

Definition 2.8 *If $\mathcal{P} \subseteq \Sigma^\omega$ is a relative liveness property of $lim(L)$ we say that $lim(L)$ satisfies $\mathcal{P}$ approximately.*

The name *approximate* satisfaction is motivated by observing that $\mathcal{P}$ is a relative liveness property of $lim(L)$ if and only if $lim(L) \cap \mathcal{P}$ is a dense set in the Cantor topology on $lim(L)$ [12, 13]. We can also give an alternative set-inclusion characterization of an approximately satisfied property which establishes the decidability of approximate satisfaction for ω-regular behaviours and properties.

Lemma 2.9 *$lim(L)$ satisfies $\mathcal{P}$ approximately if and only if*

$$pre(lim) = pre(lim(L) \cap \mathcal{P}).$$

From an intuitive point of view, approximate satisfaction and satisfaction of properties under fairness [6, 11] are closely related. But approximate satisfaction can differ from satisfaction under particular fairness concepts. One can show that a finite-state implementation of a behaviour such that all strongly fair computations of the implementation satisfy the property can always be found [12, 14]. Unfortunately this finite-state implementation can be much larger (product of the state-spaces of the behaviour and the property) than the minimal finite-state system accepting $lim(L)$ and thus is an inconvenient representation of $lim(L)$. Therefore, approximate satisfaction can be regarded as satisfaction under fairness that allows a very compact representation of a system's behaviour.

3 Abstraction

When turning to verification which is checking whether a behaviour satisfies given properties (a requirement specification), the size of the state-space of the behaviour limits the applicability of automatic verification techniques. On the other hand, a system usually performs actions which need not be considered in the verification process or which need not be distinguished from other actions respectively. Hence we can try to reduce the state-space by ignoring unimportant actions or by giving a common name to actions which need not be distinguished from one another. These two concepts are known as *action hiding* and *action renaming* respectively. The so defined type of simplification is called *abstraction*.

On the level of formal language theory, the concepts of action hiding and renaming are well-established in *alphabetic language homomorphisms* [10]. Alphabetic language homomorphisms are language homomorphisms (mappings from the Kleene-closure of an alphabet to the Kleene-closure of another alphabet which are compatible with concatenation) which take letters to letters (action renaming) or to the empty word (action hiding). They are originally defined on languages. We have to extend them to ω-languages, defining the notion of an *abstraction homomorphisms*:

Definition 3.1 *Let Σ^∞ designate $\Sigma^* \cup \Sigma^\omega$. Let Σ and Σ' be two finite sets of actions. We call $h : \Sigma^\infty \to \Sigma'^\infty$ an abstraction homomorphism if and only if the following conditions hold:*

- *If we constrain h to a mapping on letters in Σ, then we obtain a total function $h : \Sigma \to \Sigma' \cup \{\varepsilon\}$. (action renaming and hiding)*
- *If $v, w \in \Sigma^*$ and $x \in \Sigma^\omega$, then $h(vw) = h(v)h(w)$ and $h(vx) = h(v)h(x)$. (compatibility with concatenation)*
- *If we constrain h to a mapping on ω-words over Σ, then we obtain a partial function $h : \Sigma^\omega \to \Sigma'^\omega$ (no reduction of infinite sequences to finite ones).*

Note that abstraction homomorphisms are partial mappings since they are not defined on ω-words which would be mapped to finitely long words. The set of ω-words x on Σ such that $h(x)$ is defined is given by $h^{-1}(h(\Sigma^\omega))$. Since we are considering behaviours that are Eilenberg-limits of prefix-closed regular languages, we have to define the *abstraction* of a concrete behaviour in terms of Eilenberg-limits:

Definition 3.2 *Let $h : \Sigma^\infty \to \Sigma'^\infty$ be an abstraction homomorphism and let $L \subseteq \Sigma^*$ be a prefix-closed language. We define the abstraction of the concrete behaviour $lim(L)$ with respect to h to be $lim(h(L))$.*

This definition is reasonable since, for prefix-closed regular L, the two sets $lim(h(L))$ and $h(lim(L))$ are equal [12, 13, 14]. When considering approximately satisfied properties on an abstract behaviour, abstraction homomorphisms do not establish a suitable abstraction concept since they do not preserve approximately satisfied properties. *Preservation of properties* designates that a property which holds for the abstraction holds for the concrete behaviour in a corresponding way. To ensure preservation of approximately satisfied properties, an additional requirement must be satisfied by the abstraction homomorphism which is called *simplicity* of the homomorphism on the behaviour [15, 16, 17]:

Definition 3.3 *Let $h : \Sigma^\infty \to \Sigma'^\infty$ be an abstraction. Let $L \subseteq \Sigma^*$ be a partial behaviour. h is called simple on L if and only if for all $w \in \Sigma^*$ we have*

$$\forall v \in cont(h(w), h(L)) : cont(v, cont(h(w), h(L))) = cont(v, h(cont(w, L))).$$

Abstractions which are simple on a partial behaviour L establish exactly the class of abstractions which preserve relative liveness properties [12, 13]:

Theorem 3.4 *Let $L \subseteq \Sigma^*$ be a prefix-closed and regular language, let $h : \Sigma^\infty \to \Sigma'^\infty$ be an abstracting homomorphism such that $h(L)$ does not contain maximal words,*[1] *and let $\mathcal{P} \subseteq \Sigma'^\omega$ be a property. Then the condition*

$$lim(h(L)) \models_{RL} \mathcal{P} \quad \textit{if and only if} \quad lim(L) \models_{RL} h^{-1}(\mathcal{P})$$

holds if and only if h is simple on L.

According to this theorem we can derive approximately satisfied properties of the concrete behaviour by considering approximately satisfied properties of its (simple) abstraction. In the subsequent sections we will show that this can be done even without knowing the concrete behaviour completely.

[1] $w \in L$ is a maximal word in L if and only if $cont(w, L) = \{\varepsilon\}$.

4 Partial-Orders of Actions

A different way to tackle the state-space explosion problem (the problem of tremendously large state-spaces of industrial-sized specifications) is established by *partial-order methods*. If there are several different partial computations of a behaviour which are equal except for permutations of adjacent *independent* actions (independent actions "do not influence one another"; see below) then the representation of the behaviour will be reduced to one in which for each class of equivalent partial computations only a reduced number of representatives is included. Partial-order methods ignore interleavings of particular concurrent behavioural patterns and hence reduce the state-space by ignoring interim states. The key notion in the definition of partial-order methods is the *independence* of actions:

Definition 4.1 *Let Σ be a finite set of actions. A relation $\Delta \subseteq \Sigma \times \Sigma$ is called an* independence relation *of a finite-state system A (i.e. a finite automaton with only accepting states) if and only if for all a and a' in Σ, $(a, a') \in \Delta$ if and only if for all states q of A:*

- *if a is enabled in q, then if a' is taken, a will still be enabled in the successor state of q after taking a'.*
- *if a and subsequently a' is taken in state q, the successor state will be the same as if first a' and then a were taken.*

We consider two partial computations to be equivalent if and only if we can transform one into the other by permuting adjacent independent actions. The equivalence classes of this equivalence relation are called *traces*.

Definition 4.2 *Let $w \in \Sigma^*$ be a partial computation and let $\Delta \subseteq \Sigma \times \Sigma$ be an independence relation. The trace $[w]_\Delta$ according to w and Δ is the set of all partial computations w' such that w' can be constructed from w by permutations of adjacent independent actions.*

Definition 4.3 *Let A be a finite-state system and let $\Delta \subseteq \Sigma \times \Sigma$ be an independence relation. A trace system A' according to A and Δ is a finite-state system such that:*

- *For all w accepted by A, there exists w' accepted by A' such that $w \in [w']_\Delta$.*
- *For all wa accepted by A', $w \in L(A')$ and $a \in \Sigma$, there is no $b \in \Sigma$ such that $(a, b) \in \Delta$ and $wb \in L(A')$.*
- *All w' accepted by A' are also accepted by A.*

Lemma 4.4 *Let A be a finite-state system on Σ. Let $\Delta \subseteq \Sigma \times \Sigma$ be an independence relation. Then, for all wa accepted by A such that $w \in L(A)$ and $a \in \Sigma$, there exists no $b \in \Sigma$ such that $(a, b) \in \Delta$ and $wb \in L(A')$ if and only if, for all $w \in L(A)$, no $w' \in [w]_\Delta$ such that $w' \neq w$ is in $L(A)$.*

Proof "⇒": Assume there exist $v \in L(A)$ and $a, b \in \Sigma$ such that $(a, b) \in \Delta$ and $va \in L(A)$ and $vb \in L(A)$. We construct w and w' in $L(A)$ such that $w' \in [w]$. Because va and vb are both in $L(A)$, actions a and b are both enabled after the partial computation v occured. Because they are independent actions, b is still enabled after taking a and a is still enabled after taking b. Thus both, vab and vba are in $L(A)$, and $vba \in [vab]$.

"⇐": Assume that $w' \in [w]_\Delta \cap L(A)$. We show that this implies the existence of va and vb in $L(A)$ such that $v \in L(A)$ and $(a, b) \in \Delta$. Let v be the longest common prefix of w and w'. Let a and b be the actions which follow v in w and w' respectively. Because $w' \in [w]_\Delta$, a and b be must be independent, and va and vb are both in $L(A)$. □

5 Compatibility of Partial-Orders and Abstraction

When considering abstraction in practice, concrete state-spaces are in general much too large to be constructed exhaustively before collapsing them by an abstraction step. Therefore approaches are needed to construct only a part of the concrete state-space which is sufficient to compute the abstract state-space. Then, using the results sketched in the preliminaries of this paper, properties of a system can be checked without constructing its state-space exhaustively. If, in addition, simplicity of the homomorphism can be checked on the reduced concrete state-space, such an approach can also be established for properties under fairness assumptions (namely approximately satisfied properties).

We establish in this section such a result partly by combining abstraction with the concept of a *trace system*. A trace system is defined very rigorously in order to enable us to check simplicity of a homomorphism on it. Unfortunately, constructing it from a specification of a system appears to be problematic. However, without requiring to be able to check simplicity of a homomorphism on the complete behaviour by looking at the trace system, the notion of a trace system can be relaxed easily to one being constructible from a specification using usual partial-order methods [7, 18].

Definition 5.1 *Let $h : \Sigma^\infty \to \Sigma'^\infty$ be an abstraction homomorphism. An independence relation $\Delta \subseteq \Sigma \times \Sigma$ is h-compatible if and only if $(a, b) \in \Delta$ implies $h(a) = \varepsilon$ or $h(b) = \varepsilon$ or $h(a) = h(b)$. A trace system is called h-compatible if and only if its underlying independence relation is h-compatible.*

Lemma 5.2 *Let $h : \Sigma^\infty \to \Sigma'^\infty$ be an abstraction homomorphism. Let $\Delta \subseteq \Sigma \times \Sigma$ be a h-compatible independence relation. Let $w' \in \Sigma^*$ and let $w \in [w']_\Delta$. Then $h(w) = h(w')$.*

Proof Let $(w^i)_{1\leq i\leq n}$, $n \in I\!N$, be a sequence of partial computations such that $w^1 = w$, $w^n = w'$, and for all $1 \leq i \leq n-1$, w^{i+1} can be derived from w^i be permuting exactly one pair of adjacent independent actions. Let $(a,b) \in \Delta$ be such a pair. Because Δ is h-compatible, we have $h(a) = \varepsilon$ or $h(b) = \varepsilon$ or $h(a) = h(b)$. Hence, $h(ab) = h(ba)$ and consequently $h(w^i) = h(w^{i+1})$, for all $1 \leq i \leq n-1$. Thus $h(w) = h(w')$. $\Box$

Lemma 5.3 *Let $h : \Sigma^\infty \to \Sigma'^\infty$ be an abstraction homomorphism, let A be a finite state system, and let $\Delta \subseteq \Sigma \times \Sigma$ be a h-compatible independence relation. Let A' be a trace system according to A and Δ, let $w' \in L(A')$, and let $w \in L(A)$ such that $w \in [w']_\Delta$. Then*

1. $h(L(A)) = h(L(A'))$,
2. $h(cont(w, L(A))) = h(cont(w', L(A')))$,
3. $h(cont(w', L(A))) = h(cont(w', L(A')))$.

Proof

1. Because by definition of a trace system, all $w' \in L(A')$ are also in $L(A)$, we only have to show that $h(L(A)) \subseteq h(L(A'))$. Let v be in $L(A)$. Then there is v' in $L(A')$ such that $v \in [v']_\Delta$. By Lemma 5.2 $h(v) = h(v')$ which implies $h(L(A)) \subseteq h(L(A'))$.

2. Because by definition of a trace system, all $w' \in L(A')$ are also in $L(A)$, we only have to show that $h(cont(w, L(A))) \subseteq h(cont(w', L(A')))$. Let v be in $cont(w, L(A))$. Then $wv \in L(A)$. Thus there is a $u' \in L(A')$ such that $wv \in [u']$. By definition of a trace system, w' such that $w \in [w']$ is a prefix of u'. Hence $u' = w'v'$ such that $v \in [v']$. Thus $h(v) = h(v')$.

3. This is a special case of the previous (second) case (set $w' = w$).

$\Box$

Theorem 5.4 *Let A be a finite-state system on alphabet Σ. Let $h : \Sigma^\infty \to \Sigma'^\infty$ be an abstraction homomorphism. Let A' be a h-compatible trace system to A. Then h is simple on $L(A)$ if and only if h is simple on $L(A')$.*

Proof "$\Rightarrow$": Assume that h is simple on $L(A)$. Let w' be in $L(A')$. By definition, w' is in $L(A)$. Since h is simple on $L(A)$, we know that there must exist some $v \in cont(h(w'), h(L(A)))$ such that $cont(v, cont(h(w'), h(L(A))))$ is equal to $cont(v, h(cont(w', L(A))))$. By definition of a h-compatible trace system, we have $h(L(A)) = h(L(A'))$ as well as $h(cont(w', L(A))) = h(cont(w', L(A')))$. Hence,

$cont(v, cont(h(w'), h(L(A')))) = cont(v, h(cont(w', L(A'))))$. Conseqently, h is simple on $L(A')$.

"$\Leftarrow$": Assume that h is simple on $L(A')$. Let w be in $L(A)$. By definition, w is in $[w']_\Delta$ for some $w' \in L(A')$. Since h is simple on $L(A')$, there must exist some $v' \in cont(h(w'), h(L(A')))$ such that $cont(v', cont(h(w'), h(L(A'))))$ is equal to $cont(v', h(cont(w', L(A'))))$. By definition of a h-compatible trace system, we have $h(L(A)) = h(L(A'))$ and $h(cont(w, L(A))) = h(cont(w', L(A')))$. Consequently, $cont(v', cont(h(w), h(L(A)))) = cont(v', h(cont(w, L(A))))$ and h is simple on $L(A)$. □

We can combine this result with the preservation result for approximately satisfied properties (Theorem 3.4) and Lemma 5.3 and obtain finally:

Corollary 5.5 *Let A be a finite-state system on alphabet Σ. Let $h : \Sigma^\infty \to \Sigma'^\infty$ be an abstraction homomorphism. Let A' be a h-compatible trace system to A such that $h(L(A'))$ does not contain maximal words. Let $\mathcal{P} \subseteq \Sigma'^\omega$ be a property. Then the condition*

$$lim(h(L(A')))\models_{RL} \mathcal{P} \quad \textit{if and only if} \quad lim(L(A))\models_{RL} h^{-1}(\mathcal{P})$$

holds if and only if h is simple on $L(A')$.

6 Conclusion

We have shown in this paper (Corollary 5.5) that we can use a reduced state-space (a partial-order representation) of a specification to check properties even under fairness without constructing the complete state-space of the specification exhaustively. This verification includes an interim abstraction step for which it is shown that the complete and reduced behaviour of the specification lead to the same abstract behaviour. The key concept for this approach is *abstraction-compatibility* of the independence relation on actions.

In addition checking simplicity of the abstraction, which is crucial for the preservation of so called approximately satisfied properties, can also be checked on the reduced behaviour on behalf of the complete one. However this result is limited by observing that it is not certain whether a trace system as defined in this paper is constructible from a given specification. The problem lies in the fact that each trace is represented only by a single representative (according to Lemma 4.4) which may be a too strict definition. Without requiring simplicity of the abstraction, the results of this paper can be relaxed easily to a definitely constructive definition of a trace system. Since approximate satisfaction, which is an abstract notion of property satisfaction under fairness, is a satisfaction relation of practical necessity for the verification of co-operating systems, it is a topic for further study whether an algorithm for the construction of trace systems from specification can be found.

In case of a negative answer to this question, a relaxed definition of a trace system based on a usual partial-order construction has to be found which still enables checking simplicity of an abstraction on the (not constructed) complete behaviour of a specification.

Consequently, this paper presents the border conditions and their implications for the construction of a simple (i.e. approximately property preserving) abstraction of a state-space from its reduced, partial-order-based representation.

References

[1] M. Abadi and L. Lamport. The existence of refinement mappings. SRC Report 29, DEC System Research Center, July 1988.

[2] M. Abadi and L. Lamport. Composing specifications. SRC Report 66, DEC System Research Center, October 1990.

[3] B. Alpern and F. B. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181–185, October 1985.

[4] R. Alur and T. A. Henzinger. Local liveness for compositional modeling of fair reactive systems. In P. Wolper, editor, *Computer Aided Verification (CAV) '95*, volume 939 of *Lecture Notes in Computer Scinece*, pages 166–179. Springer, 1995.

[5] J. Berstel. *Transductions and Context-Free Languages.* Studienbücher Informatik. Teubner Verlag, Stuttgart, first edition, 1979.

[6] N. Francez. *Fairness.* Springer Verlag, New York, first edition, 1986.

[7] P. Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems – An Approach to the State-Explosion Problem.* PhD thesis, Université de Liège, Liège, Belgium, 1995.

[8] M. A. Harrison. *Introduction to Formal Language Theory.* Addison-Wesley, Reading, Mass., first edition, 1978.

[9] T. A. Henzinger. Sooner is safer than later. *Information Processing Letters*, 43:135–141, 1992.

[10] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, Reading, Mass., first edition, 1979.

[11] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems—Specification.* Springer Verlag, New York, first edition, 1992.

[12] U. Nitsche. *Verification of Co-Operating Systems and Behaviour Abstraction.* PhD thesis, University of Frankfurt, Germany, 1998.

[13] U. Nitsche and P. Ochsenschläger. Approximately satisfied properties of systems and simple language homomorphisms. *Information Processing Letters*, 60:201–206, 1996.

[14] U. Nitsche and P. Wolper. Relative liveness and behavior abstraction (extended abstract). In *Proceedings of the 16th ACM Symposium on Principles of Distributed Computing (PODC'97)*, pages 45–52, Santa Barbara, CA, 1997.

[15] P. Ochsenschläger. Verifikation kooperierender Systeme mittels schlichter Homomorphismen. Arbeitspapiere der GMD 688, Gesellschaft für Mathematik und Datenverarbeitung (GMD), Darmstadt, Oktober 1992.

[16] P. Ochsenschläger. Verification of cooperating systems by simple homomorphisms using the product net machine. In J. Desel, A. Oberweis, and W. Reisig, editors, *Workshop: Algorithmen und Werkzeuge für Petrinetze*, pages 48–53. Humboldt Universität Berlin, 1994.

[17] P. Ochsenschläger. Compositional verification of cooperating systems using simple homomorphisms. In J. Desel, H. Fleischhack, A. Oberweis, and M. Sonnenschein, editors, *Workshop: Algorithmen und Werkzeuge für Petrinetze*, pages 8–13. Universität Oldenburg, 1995.

[18] P. Wolper and P. Godefroid. Partial-order methods for temporal verification. In E. Best, editor, *CONCUR'93*, volume 715 of *Lecture Notes in Computer Science*, pages 233–246. Springer Verlag, 1993.

Characterizing bisimilarity of value-passing processes with context-free control

Paweł Pączkowski

Institute of Mathematics
University of Gdańsk
ul. Wita Stwosza 57
80–952 Gdańsk, Poland

matpmp@univ.gda.pl

June 1998

1 Introduction

We consider a value-passing version of context-free processes, where process behaviours depend on a global state of data variables. Accordingly, the semantics and bisimulation depend on valuations of data variables. In this work we search for a formula characterizing those valuations of data variables for which two given processes are bisimilar. We extend the results of [2, 4], where regular processes were dealt with. The techniques used in these references cannot be directly applied to processes with context-free control. Here we exploit a tableau based decision procedure for bisimilarity for context free processes [1]. This note presents an ongoing work.

2 Processes

The class of processes we consider is obtained by extending normed context-free processes, as defined in [1], with value passing primitives.

Let $X, Y, Z \ldots$ be a finite set of process variables. A context free processes is defined by a finite set of equations Δ of the form

$$X_i = \sum_{j=1}^{n_i} a_{ij}\beta_{ij}, \qquad 1 \leq i \leq m \tag{1}$$

where X_i are process variables, a_{ij} are actions described below and β_{ij} are process variable sequences of length at most two (that is, we take processes expressed in Greibach normal

form). For every equation Δ we assume some finite set of data variables ranged over by x. The actions a_{ij} can have two forms:

$$(b\,,\, c?x\,,\, s) \qquad \text{or} \qquad (b\,,\, c!e\,,\, s) \tag{2}$$

where b is a boolean expression, c is a channel name drawn from a fixed set of channel names, x is a data variable, e is an expression from a presupposed set of data expressions, and s is a simultaneous assignment:

$$(x_1, \ldots, x_k) := (e_1, \ldots, e_k)$$

Intuitively, an action $(b\,,\, c?x\,,\, s)$ should be interpreted as an input of a value from channel c, under the condition that boolean b holds (before the new value of x has been received), after which the data variables are updated by performing assignment s. The output action $(b\,,\, c!e\,,\, s)$ differs in that the value of expression e is output to channel c.

Thus, so defined processes have a global state represented by a valuation of data variables. The actions can modify the state by performing inputs and assignments; the output values depend on the current valuation of data variables. The set of equations Δ describes the context-free control of the process execution.

Formally, the semantics of processes is defined by the following two-step procedure. Given a set of equations Δ, we define first transitions of the form

$$\alpha \xrightarrow{\ a\ } \beta$$

where α, β are sequences of process variables and a is an action as described by (2). Such transitions can be defined in a standard manner: we follow [1] treating our actions just as symbols, disregarding their structure. We call these transitions *symbolic*. Next, we interpret the symbolic transitions with respect to valuations of data variables to derive *semantic transitions* of the form

$$\langle \sigma, \alpha \rangle \xrightarrow{\ c?v\ } \langle \rho, \beta \rangle \qquad \text{or} \qquad \langle \sigma, \alpha \rangle \xrightarrow{\ c!v\ } \langle \rho, \beta \rangle$$

where σ, ρ are valuations of data variables, i.e. functions from data variables to a presupposed set of data values, c is a channel name and v ranges over data values. The actions labelling the semantic transitions, $c?v$ or $c!v$, represent, respectively, an input of value v from channel c and output of of value v on channel c.

The following two rules are used to derive semantic transitions. We use the notation $\sigma[s]$ for a valuation obtained from σ by performing assignment s and we denote by $[\![e\sigma]\!]$ the value of expression e under valuation σ.

$$\frac{\alpha \xrightarrow{\ b\,,\, c!e\,,\, s\ } \beta \qquad \sigma \models b}{\langle \sigma, \alpha \rangle \xrightarrow{\ c![\![e\sigma]\!]\ } \langle \sigma[s], \beta \rangle} \tag{3}$$

$$\frac{\alpha \xrightarrow{\ b\,,\, c?x\,,\, s\ } \beta \qquad \sigma \models b}{\langle \sigma, \alpha \rangle \xrightarrow{\ c?v\ } \langle \sigma[x := v][s], \beta \rangle}, \quad \text{for any value } v \tag{4}$$

3 Characterizing Bisimilarity

We adopt the standard definition of (strong) bisimulation equivalence between labelled transition systems (see e.g. [3]), denoting it by $\sim$.

Let Δ and Δ' be two sets of equations defining value-passing context-free processes. Assume that process and data variables of Δ are disjoint from process and data variables of Δ'. This can be always achieved by renaming variables. Let X and X' be process variables appearing, respectively, in Δ and Δ'. The problem we consider is to characterize those valuations σ of data variables for which $\langle \sigma, X \rangle$ is bisimilar to $\langle \sigma, X' \rangle$. More precisely, we look for a formula $B_{X,X'}$ of the first order logic built on top of assumed sets booleans and data expressions such that

$$\sigma \models B_{X,X'} \quad \text{iff} \quad \langle \sigma, X \rangle \sim \langle \sigma, X' \rangle.$$

In [2, 4], formulas characterizing bisimilarity in the sense above have been studied for regular processes. Such formulas were found as solutions to systems of equivalences in a first order logic extended with explicit substitutions. A system of equivalences consists of a finite set of equivalences of the form

$$P_i \;\Leftrightarrow\; \Phi_i(P_1, \ldots, P_m), \qquad\qquad i = 1 \ldots m \tag{5}$$

where P_i are predicate variables and $\Phi_i(P_1, \ldots, P_m)$ are formulas of a first order logic extended with explicit substitutions. The techniques used in [2, 4] do not apply immediately to processes with context-free control as infinite sets of equivalences could arise.

In this work we observe that the tableau decision method for normed context free processes described in [1] can be successfully exploited to extend the results of [2, 4].

The tableau decision method builds a tableau composed of rules of the shape

$$\frac{E\alpha = E'\alpha'}{E_1\alpha_1 = E'_1\alpha'_1 \quad \ldots \quad E_k\alpha_k = E'_k\alpha'_k}$$

where E (possibly decorated) is either empty or stands for a sum of the form $\sum_{j=1}^{n} a_i\beta_i$.

Following the approach of [4] we associate a predicate variable $B_{\alpha,\alpha'}$ with every equation $\alpha = \alpha'$ (whose component E is empty) appearing in a tableau and encode the tableau as a set of equivalences $EQ(\Delta, \Delta')$ whose predicate variables are $B_{\alpha,\alpha'}$. The crucial point is that tableaux are guaranteed to be finite for normed context-free processes, therefore the systems of equivalences we construct are also finite.

The following Proposition holds for the case of *deterministic* normed processes with context-free control, i.e. such that for every i in the defining equation (1), in the set of actions $\{a_{ij} \mid 1 \leq j \leq n_i\}$, channels used for inputs are mutually different and channels used for outputs are mutually different.

Proposition 1 *Let Δ, Δ' be two sets of equations that define deterministic normed value-passing processes with context-free control. One can effectively construct a set of equivalences $EQ(\Delta, \Delta')$ of first order formulas with explicit substitutions such that any set of*

predicates $\{B_{\alpha,\alpha'}\}$ *which is a solution to* $EQ(\Delta, \Delta')$ *satisfies*

$$\sigma \models B_{\alpha,\alpha'} \quad \textit{iff} \quad \langle \sigma, \alpha \rangle \sim \langle \sigma, \alpha' \rangle$$

□

We believe that Proposition above holds also for the nondeterministic case.

References

[1] H. Hüttel and C. Stirling, Actions speak louder than words: proving bisimilarity for context-free processes, in: *Proceedings of LICS 91*, pp. 376–386, IEEE computer Society Press, 1991.

[2] H. Lin, Symbolic transition graph with assignment, in: *Proceedings of CONCUR'96*, Springer-Verlag LNSC, 1996.

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

[4] P. Pączkowski, Characterizing bisimilarity of value-passing parametrised processes, in: B. Steffen and T. Margaria (Eds.), INFINITY, International Workshop on Verification of Infinite State Systems, *Electronic Notes of Theoretical Computer Science* 5, 1997.

Hardness results for weak bisimilarity of simple process algebras

Jitka Stříbrná
LFCS, Department of Computer Science
University of Edinburgh, UK
email: js@dcs.ed.ac.uk

We do not know whether weak bisimilarity (denoted $\approx$) is decidable for general BPA and BPP but we may try to estimate what would be a least complexity of a decision procedure that might exist. That is achieved by taking problems complete for some complexity classes and reducing them to $\approx$. In this way we will show that the problem of deciding weak bisimilarity would be NP-hard for BPP, and PSPACE-hard for BPA.

1 Background

There has been much effort devoted to the study of decidability of various bisimulation equivalences for many process calculi. The majority of the results concern strong bisimilarity which was shown to be decidable for the two classes of Basic process algebras (BPA) [2] and Basic parallel process algebras (BPPA) [1] that we will be considering in this paper.

However, we want to focus our attention on a more interesting notion of observation that is weak bisimilarity. So far, the decidability of weak bisimilarity for the subclass of *totally normed* BPP and BPA was established in [5], and a semidecision procedure for weak bisimilarity of BPP was manifested in [3]. The question for general BPA and BPP remains open, however we may at least try to establish some lower bounds on a decision procedure that might exist.

Decidability of bisimulation equivalences so far seems to be consistent with polynomial-time decision procedures. Polynomial algorithms deciding strong bisimilarity for normed BPP and BPA were manifested in [6], [7], [8], and even though there is no polynomial decision procedure for the class of all BPP and BPA, there is no lower bound that contradicts its existence. It would be a weak negative result to show that weak bisimilarity cannot be decidable in polynomial time. To our best knowledge even this weak result hasn't been proved yet, however in this paper we provide a strong evidence that the problem of deciding weak bisimilarity cannot be solved in polynomial time. More specifically, we will show that for weak bisimilarity and (totally normed) BPP and BPA the decision problem is at least NP-hard and for general BPA, at least PSPACE-hard.

Before we proceed to demonstrate these claims we will recall the necessary notions from process algebra and computational complexity.

We will start with the process algebras. We presuppose a fixed set of *actions* $Act = \{a, b, c, \ldots\}$ that contains a special *silent* action τ, and a finite set of process variables or atoms $\Sigma = \{X_1, \ldots, X_n\}$. Then we say that a *basic process algebra* or *BPA* is a pair (Σ^*, Δ) where Σ^* is the free monoid generated by Σ, and $\Delta = \{X \stackrel{\alpha}{\longrightarrow} P \mid X \in \Sigma, P \in \Sigma^*, \alpha \in Act\}$ is a finite set of transitions. We call words from Σ^* BPA-processes. The transition rules of Δ determine a transition relation on general BPA-processes in this way:

$$XQ \stackrel{\alpha}{\longrightarrow} PQ \text{ if there is a rule } X \stackrel{\alpha}{\longrightarrow} P \text{ in } \Delta$$

A *basic parallel process algebra* or *BPPA* is a pair $(\hat{\Sigma}, \Delta)$, where $\hat{\Sigma} = \{X_1^{i_1} \ldots X_n^{i_n} \mid i_1, \ldots, i_n \in \mathbb{N}\}$ is the commutative algebra generated by Σ, and $\Delta = \{X \stackrel{\alpha}{\longrightarrow} P \mid X \in \Sigma, P \in \hat{\Sigma}, \alpha \in Act\}$ is a finite set of transitions. We call multisets from $\hat{\Sigma}$ basic parallel processes or BPP. In the algebra $\hat{\Sigma}$ there is a natural operation which we shall call *parallel composition* and denote with $\|$:

$$P \parallel Q = X_1^{i_1+j_1} \ldots X_n^{i_n+j_n}, \text{ where } P \equiv X_1^{i_1} \ldots X_n^{i_n} \text{ and } Q \equiv X_1^{j_1} \ldots X_n^{j_n}.$$

We can extend the rules of Δ to all BPP in the obvious way:

$$P \parallel X \parallel Q \stackrel{\alpha}{\longrightarrow} P \parallel R \parallel Q \text{ if there is a rule } X \stackrel{\alpha}{\longrightarrow} R \text{ in } \Delta$$

For both BPA and BPPA we use capital letters X, Y to range over process variables and P, Q, R to range over processes.

A process variable X is said to be *totally normed* if there is no transition $X \stackrel{\tau}{\longrightarrow} \epsilon$ and if there exists a derivation $X \stackrel{s}{\Longrightarrow} \epsilon$ for some non-empty sequence of actions s. A process algebra is totally normed if all its variables are totally normed.

Now we will introduce the concepts of strong and weak bisimilarity. We say that a binary relation $\mathcal{R}$ over pairs of processes is a *strong bisimulation relation* if the following condition holds for every pair (P, Q) from $\mathcal{R}$ and every action α from Act:

- for every $P \stackrel{\alpha}{\longrightarrow} P'$ there exists $Q \stackrel{\alpha}{\longrightarrow} Q'$ so that $(P', Q') \in \mathcal{R}$
- for every $Q \stackrel{\alpha}{\longrightarrow} Q'$ there exists $P \stackrel{\alpha}{\longrightarrow} P'$ so that $(P', Q') \in \mathcal{R}$

Two processes P and Q are strongly bisimilar if there exists a strong bisimulation relation containing the pair (P, Q). The union of all bisimulation relations gives rise to the maximal bisimulation which is denoted by $\sim$.

A *weak bisimulation relation* is defined analogously as a strong bisimulation relation with the single transitions $\stackrel{\alpha}{\longrightarrow}$ being replaced with $\stackrel{\alpha}{\Longrightarrow}$, where $\stackrel{\alpha}{\Longrightarrow}$ is an abbreviation of $(\stackrel{\tau}{\longrightarrow})^* \stackrel{\alpha}{\longrightarrow} (\stackrel{\tau}{\longrightarrow})^*$ in case of $\alpha \neq \tau$ and $(\stackrel{\tau}{\longrightarrow})^*$ in case of $\alpha = \tau$. Then we say that processes P and Q are weakly bisimilar if there exists a weak bisimulation relation containing the pair (P, Q). There also exists a maximal weak bisimulation relation which is obtained as a union of all weak bisimulation relations and it is denoted by $\approx$.

Now we will recall some notions from computational complexity (consult [4], [9] and [11] for more details). Informally, we say that a problem P is $\mathcal{C}$-hard for some complexity class $\mathcal{C}$ if to solve P is as difficult as to solve any problem from $\mathcal{C}$. If on top of that we know that the complexity of solving P is $\mathcal{C}$, in other words P belongs to $\mathcal{C}$, we say that P is $\mathcal{C}$-complete. Often when we try to estimate a lower bound on a complexity of some problem P we transform another problem P' to P where we know the complexity of P' already. For this purpose we use the concept of *reduction*.

Assume two languages L_1 over some alphabet Σ_1 and L_2 over an alphabet Σ_2. A *reduction* from L_1 to L_2 is a function f from Σ_1^* to Σ_2^* such that

$$\text{for all } w \in \Sigma_1^*, \qquad w \in L_1 \iff f(w) \in L_2.$$

When reducing one problem to another we need a function that can be efficiently computed. As we will see later, it suffices for the reduction f to be *polynomial time*, that is we can construct a polynomial-time bound Turing machine that computes the function f. We will denote the fact that L_1 is polynomial-time reducible to L_2 by $L_1 \preccurlyeq L_2$.

We say that a language L is *complete* for a class $\mathcal{C}$ (with respect to polynomial-time reduction) if L is in $\mathcal{C}$ and every language in $\mathcal{C}$ is reducible to L. A language L is *hard* for $\mathcal{C}$ (wrt polynomial-time reduction) if every language in $\mathcal{C}$ is reducible to L, but L is not necessarily in $\mathcal{C}$. The complexity classes that we will be dealing with are the classes NP and PSPACE. The following theorem [9] confirms that polynomial-time reduction is suitable for our purposes.

Lemma 1 *If $L' \in \mathcal{P}$ and $L \preccurlyeq L'$ then also $L \in \mathcal{P}$, where $\mathcal{P}$ is the class of problems that can be solved in polynomial time.*

2 Weak bisimilarity of BPP is NP-hard

There is a plethora of known problems that fall into the class of NP-complete problems. We need to choose a problem that is in some way related to the principle of bisimilarity. Clearly, any two processes that are bisimilar must be capable of performing sequences of actions of identical lengths therefore we will reduce a 'counting' problem to weak bisimilarity.

One problem that suits our purpose is KNAPSACK which reads as follows: given a sequence of values $m_1, m_2, \ldots, m_n$ and a total t, we want to find out whether we can choose a subsequence $m_{i_1}, \ldots, m_{i_k}$ that adds up to t. That brings us to the idea of having two processes, one simulating the total t by being defined as a trace of the length of t, and the other simulating the choices of subsequences of $m_1, m_2, \ldots, m_n$ and hence being equivalent to a tree whose branches correspond to traces of lengths specified by the individual subsequences.

KNAPSACK is known to be NP-complete (cf. [4], [11]) and we will demonstrate a polynomial-time reduction to weak bisimilarity of basic parallel processes. As we have explained earlier that will establish that the problem of deciding $\approx$ for BPP is at least NP-hard. The formal definition of KNAPSACK is as follows:

Definition 2 KNAPSACK *is the following problem:*

Instance: $t, m_1, m_2, \ldots, m_n \in \mathbb{N}$
Question: $\exists i_1, i_2, \ldots, i_n \in \{0,1\}.\ \sum_{j=1}^{n} i_j m_j = t$*?*

The fact that this simple problem is NP-complete is due to the fact that there may occur arbitrarily large numbers on the input. If we consider the input encoded in unary then we can find an algorithm that solves KNAPSACK in time polynomial in size of the input. The possibility of arbitrarily large values of m_i or t will require a trick in the definition of processes so that we remain within the limits of polynomial-time reduction. We will now proceed to demonstrate a polynomial time many-one reduction. of KNAPSACK to weak bisimilarity of BPP.

Lemma 3 KNAPSACK $\preceq \approx$.

Proof: Let $t, m_1, m_2, \ldots, m_n \in \mathbb{N}$ be an instance of KNAPSACK. We will demonstrate two basic parallel processes P and Q such that there exist $i_1, i_2, \ldots, i_n \in \{0,1\}$ with $\sum_j i_j m_j = t$ if and only if $P \approx Q$. Following the afore mentioned idea, the process P will simulate branching defined by individual subsequences, and the process Q will simulate the trace of length t. For the purpose of counting we will use a single visible action a that will help us to test if there is a branch in the tree defined by P of length t, ie. equivalent with Q.

For each m_j, resp. t, we will introduce a process variable M_j, resp. T, that will be able to perform exactly a sequence of $\stackrel{a}{\Longrightarrow}$ transitions of length m_j, resp. t. The process P then will be capable of generating any subset of $\{M_1, \ldots, M_n\}$ whereas the process Q will be able to evolve into T. Finally we will demonstrate that P is weakly bisimilar to Q if and only if the answer to the respective instance is yes. Now we will present the transition rules that define the process variables P and Q:

$$P \stackrel{\tau}{\longrightarrow} P_1 \parallel \ldots \parallel P_n \qquad Q \stackrel{\tau}{\longrightarrow} P \qquad P_j \stackrel{\tau}{\longrightarrow} M_j,\ \ j = 1, \ldots, n$$
$$Q \stackrel{\tau}{\longrightarrow} T \qquad P_j \stackrel{\tau}{\longrightarrow} \epsilon,\ \ j = 1, \ldots, n$$

In order to complete the definitions of P and Q we have to define process variables M_j and T. Our only concern is that the resulting reduction is polynomial time hence we have to use a little trick in the definition. We define a sequence of variables $S_0, S_1, \ldots, S_k$ in this way: $S_0 \stackrel{a}{\longrightarrow} \epsilon$, $S_{i+1} \stackrel{\tau}{\longrightarrow} S_i \parallel S_i$ for $i < k$, where k is taken to be $\lfloor \log(\max\{t, m_1, \ldots, m_n\}) \rfloor$. Thus we have obtained variables such that $S_i \approx a^{2^i}$, where we use the expression a^m in the obvious meaning. Now we can define $T \stackrel{\tau}{\longrightarrow} S_k^{e_k} \parallel \ldots \parallel S_1^{e_1} \parallel S_0^{e_0}$ where $e_k \ldots e_1 e_0$ is the binary encoding of t. The variables $M_0, \ldots, M_n$ are defined in a similar fashion: $M_j \stackrel{\tau}{\longrightarrow} S_k^{e_{kj}} \parallel \ldots \parallel S_1^{e_{1j}} \parallel S_0^{e_{0j}}$, where $e_{kj} \ldots e_{0j}$ is the binary encoding of m_j, for $j = 1, \ldots, n$. To summarise, we only need $k+1$ extra variables in order to define the processes T and M_j.

It is easily seen from the construction that P can only perform sequences of $\stackrel{a}{\Longrightarrow}$ transitions of length $\sum_j i_j m_j$ for some $i_1, i_2, \ldots, i_n \in \{0,1\}$. Therefore if this sum never adds up to t the process Q can become T and thus force non-bisimilarity with P. On the other

hand, if there exist $i_1, i_2, \ldots, i_n \in \{0,1\}$ such that $\sum_j i_j m_j = t$ the process P will generate the composition $M_1^{i_1} \parallel \ldots \parallel M_n^{i_n}$ as an answer to the move $Q \stackrel{\tau}{\longrightarrow} T$ and preserve weak bisimilarity. If it is P that takes the initiative then the process Q simply makes use of the rule $Q \stackrel{\tau}{\longrightarrow} P$ and then copies any move of P. $\square$

It is quite easy to verify that the actual reduction can be carried out in polynomial time, however we will not present the full details of the reduction here. We can finally conclude with the following theorem:

Theorem 4 *The decidability of weak bisimilarity of basic parallel processes is NP-hard.*

We have to remark that the result we have just presented seems rather unimportant since we do not know whether weak bisimilarity of BPP is decidable at all. We can however adjust the reduction for the subclass of totally normed BPP. That means we have to get rid of zero-normed processes that are present in the process algebra constructed above. The problematic processes are P_i since they have at their disposal the transition rules $P_i \stackrel{\tau}{\longrightarrow} \epsilon$. However, we can change the instance of KNAPSACK slightly in order to dispose of such processes. We can consider the instance $\sum i_j m_j + \sum m_j = t + \sum m_j$ to which the answer is 'yes' if and only if the answer to the original instance is 'yes'. That means a shift in the required values of i_j from $i_j \in \{0,1\}$ to $i_j \in \{1,2\}$ which is expressed in terms of the transitions as $P_i \stackrel{\tau}{\longrightarrow} M_i \parallel M_i$, $P_i \stackrel{\tau}{\longrightarrow} M_i$, and we also need to alter the definition of Q to $Q \stackrel{\tau}{\longrightarrow} T \parallel M_1 \parallel \ldots \parallel M_n$. This change doesn't have any impact on the size of the reduction as it can still be done in polynomial time. Hence we have removed all zero-norm processes and we can assert that $\approx$ is NP-hard even for totally normed processes.

Theorem 5 *The decidability of weak bisimilarity of totally normed basic parallel processes is NP-hard.*

3 Weak bisimulation of BPA is PSPACE-hard

When we look carefully at the reduction of KNAPSACK onto weak bisimulation of BPP we can see that it could be easily modified into a reduction to weak bisimulation of BPA. There are these transition rules which contain parallel composition - the definitions of P and each M_j and T (and the auxiliary variables S_i). However, in the whole process algebra there is only a single visible action which is a and, moreover, the process variables $M_1, \ldots, M_n$ and T each determine a single string of $\stackrel{a}{\Longrightarrow}$ moves with no available branching. Therefore it doesn't make any difference to compose such processes in the sequential or parallel fashion. The final improvement when we remove all zero-norm processes also works for BPA and hence we come to the conclusion that deciding weak bisimilarity for totally normed BPA is NP-hard.

Sequential composition, however, enables us to go even further. With the sequential structure of BPA-processes we are able to encode finite automata and hence achieve a

stronger result. We will use the *totality problem for finite automata* TOT which is PSPACE-complete and construct a polynomial time reduction onto weak bisimilarity of BPA. Thus we will show that this problem is at least PSPACE-hard. First we will define the totality problem for finite automata:

Definition 6 TOT *is the following problem:*

Instance: A nondeterministic finite automaton $\mathcal{A}$ over some alphabet Σ.
Question: Is $L(\mathcal{A})$, the language accepted by $\mathcal{A}$, equal to the total language Σ^*?

This problem is PSPACE-complete even a two-letter alphabet (cf. [4]) hence in the following we will assume that $\Sigma = \{a, b\}$. We will in fact demonstrate a linear time reduction of TOT onto weak bisimilarity of BPA.

Theorem 7 TOT $\preccurlyeq \approx$.

Now we will explain the main idea behind the reduction. We presuppose a nondeterministic finite automaton $\mathcal{A} = (\Sigma, \mathcal{Q}, \delta, q_0, \{q_k\})$, where $\Sigma = \{a, b\}$ is the input alphabet, $\mathcal{Q} = \{q_0, q_1, \ldots, q_k\}$ is the set of states with q_0 being the initial and q_k the final states, and $\delta : \mathcal{Q} \times \Sigma \to 2^{\mathcal{Q}}$ is the transition function. We will write $(q_i, x) \longmapsto q_j$ to express the fact that the state q_j belongs to the set $\delta(q_i, x)$ with x being either a or b. Also note that wlog we can assume a single final state q_k.

We will simulate words over $\{a, b\}$ by introducing two variables A and B such that A can only perform the transition $A \xrightarrow{a} \epsilon$ and B can only perform the transition $B \xrightarrow{b} \epsilon$. Then any process over A and B will determine a single word from the alphabet Σ. To simulate the total language Σ^* we will introduce a process P that will be capable of producing any string of atoms A and B. Next we would like to define a process Q that can generate all strings from $L(\mathcal{A})$. However since we are only allowed the leftmost derivation in case of BPA-processes, we will define a process that generates exactly all the reverse words from $L(\mathcal{A})$. Still, we will be able to show that $L(\mathcal{A}) = \Sigma^*$ if and only if $P \approx Q$.

For each state $q_0, q_1, \ldots, q_k$ of the automaton $\mathcal{A}$ we define a variable $Q_0, Q_1, \ldots, Q_k$ using the following rules:

if $(q_i, a) \longmapsto q_j$ then $Q_i \xrightarrow{\tau} Q_j A$, and if $(q_i, b) \longmapsto q_j$ then $Q_i \xrightarrow{\tau} Q_j B$.

For the variable Q_k that corresponds to the final state q_k we add a special rule $Q_k \xrightarrow{s} \epsilon$, where s is a special *initial* action. The purpose of s is to synchronise the two processes P and Q. Finally, we put $Q = Q_0$. It is quite straightforward to observe that a word w is in $L(\mathcal{A})$ if and only if we can via a τ sequence from the variable Q generate the process $Q_k R$ where $R \in \{A, B\}^*$ and there is a unique transition sequence $R \xrightarrow{\bar{w}} \epsilon$ with $\bar{w}$ being the reverse of w.

To complete the reduction it remains to define the process P whose task is to be able to generate all strings from $\{A, B\}^*$ and simulate the process Q, and the variables A, resp. B, that simulate the letters a, resp. b.

$P \stackrel{\tau}{\longrightarrow} QT$ $\quad P \stackrel{\tau}{\longrightarrow} PA$ $\quad A \stackrel{a}{\longrightarrow} \epsilon$ $\quad T \stackrel{\tau}{\longrightarrow} T$

$P \stackrel{s}{\longrightarrow} \epsilon$ $\quad P \stackrel{\tau}{\longrightarrow} PB$ $\quad B \stackrel{b}{\longrightarrow} \epsilon$

For technical reasons we need a process that will block any sequence of variables that the process P may have generated. The process T forms such a block since it is defined as a τ loop and therefore it is clear that $TR \approx \epsilon$ for any process R. The presence of T in the algebra means that the algebra fails to be totally normed which opens the question about the complexity of weak bisimilarity for totally normed processes. The final step is to show the correctness of the reduction.

Theorem 8 *Assume a given automaton $\mathcal{A}$ and P and Q defined as above. Then $L(\mathcal{A}) = \{a,b\}^*$ iff $P \approx Q$.*

Proof: One implication is straightforward. Assume that $L(\mathcal{A}) \neq \{a,b\}^*$ and let $w \in \{a,b\}^* \setminus L(\mathcal{A})$. Since P is constructed to generate all strings of a and b it can produce a sequence of variables capable of performing the word sw^R. However, as Q simulates the automaton $\mathcal{A}$ it cannot produce this string and thus $P \not\approx Q$.

In order to show the other direction we need to analyze the moves of P and Q. The idea is that P will wait for Q to make a move and then respond by doing $P \stackrel{\tau}{\longrightarrow} QT$ which blocks anything which P may have generated in the meantime. Clearly $Q \approx QTR$ for any process R because we can never get past T. On the other hand, Q has to respond only when P decides to generate a sequence of A's and B's and then disappear. Hence the responses of Q are:

1. $P \stackrel{\tau}{\Longrightarrow} PR, R \in \{A,B\}^*$ - in this case Q does the empty sequence $Q \stackrel{\epsilon}{\Longrightarrow} Q$
2. $P \stackrel{\tau}{\Longrightarrow} QTR, R \in \{A,B\}^*$ - in this case $Q \approx QTR$ and hence again the response is $Q \stackrel{\epsilon}{\Longrightarrow} Q$
3. $P \stackrel{s}{\Longrightarrow} R, R \in \{A,B\}^*$ - since Q can generate all strings over the alphabet $\{A,B\}$ it will be able to generate the process R via $\stackrel{s}{\Longrightarrow}$.

Although this analysis is quite informal we could easily construct a binary relation as a union of three binary relations, each corresponding to one case of the above analysis, and verify that it is a weak bisimulation relation. That only requires to test that it is closed under expansion with $\stackrel{\alpha}{\longrightarrow}$ which would again follow the structure of the proof. Therefore we have P and Q weakly bisimilar. Finally, we can conclude that the automaton generates the total language if and only if the processes P and Q are weakly bisimilar. $\Box$

It is not difficult to check that this construction can be done in time linear in the size of the input automaton as for each possible transition of the input automaton we construct a single rule of the process algebra and also the number of states only increases by a constant. Therefore we can conclude that the problem of deciding weak bisimilarity for BPA is PSPACE-hard.

4 Conclusion

We cannot draw any strong conclusions from the results presented in this paper. Although weak bisimilarity seems a rather complex notion and hence we might hope to prove that if indeed it was decidable for BPA and BPP, the complexity of a decision procedure would be rather high, there seem to be obstacles that prevent us from doing so. The idea of the KNAPSACK reduction to weak bisimilarity of BPP may be applied to more complex problems; one of them might be the verification of Presburger sentences which is known to require at least doubly exponential space.

To conclude this paper, we can draw the following comparisons: for strong bisimulation and normed BPP and BPA there exist polynomial decision procedures. When we move to totally normed BPP and BPA, for which weak bisimulation is decidable, the problem of deciding weak bisimilarity is NP-hard (which is to be expected since we are dealing with a more intricate notion). For general BPA-processes, if weak bisimilarity was decidable then the decision problem would be PSPACE-hard.

Acknowledgement

I would like to thank my advisor Mark Jerrum for suggesting this line of research and for helpful discussions throughout my work on the results presented in this paper.

References

[1] Christensen S., Hirshfeld Y. and Moller F. *Bisimulation Equivalence is Decidable for Basic Parallel Processes*, in Proceedings of CONCUR 93, LNCS 715, 143–157, 1993.

[2] Christensen S., Hüttel H. and Stirling C. *Bisimulation equivalence is decidable for all context-free processes*, in Proceedings of CONCUR 92, LNCS 630, 138–147, 1992.

[3] Esparza J. *Petri nets, commutative context-free grammars and basic parallel processes.* In Fundamentals of Computation Theory 95, LNCS 965, Springer Verlag, 221–232, 1995.

[4] Garey M.R. and Johnson D.S. *Computers and Intractability: A Guide to the Theory of NP-completness.* Freeman, San Francisco, 1979.

[5] Hirshfeld Y. *Bisimulation Trees and the Decidability of Weak Bisimulations.* Draft of a paper.

[6] Hirshfeld Y., Jerrum M. and Moller F. *A polynomial-time algorithm for deciding bisimulation equivalence of normed Basic Parallel Processes.* In LFCS Report Series, University of Edinburgh, 1994.

[7] Hirshfeld Y., Jerrum M. and Moller F. *A polynomial-time algorithm for deciding bisimulation equivalence of normed Basic Parallel Processes.* In Math. Struct. in Comp. Science 6, 251–259, Cambridge University Press, 1996.

[8] Hirshfeld Y., Jerrum M. and Moller F. *A polynomial algorithm for deciding bisimilarity of normed context-free processes.* In LFCS Report Series, University of Edinburgh, 1994.

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

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

[11] Papadimitriou C.H. *Computational Complexity.* Addison-Wesley, 1994.

Place Bisimulation Equivalences for Design of Concurrent Systems *

Igor V. Tarasyuk

A.P. Ershov Institute of Informatics Systems,
6, Acad. Lavrentiev ave., Novosibirsk, 630090, Russia
Fax: +7 3832 32 34 94
E-mail: itar@iis.nsk.su

Abstract

In this paper, we supplement the set of basic and back-forth behavioural equivalences for Petri nets considered in [11] by place bisimulation ones. The relationships of all the equivalence notions are examined, and their preservation by refinements is investigated to find out which of these relations may be used in top-down design. It is demonstrated that the place bisimulation equivalences may be used for the compositional and history preserving reduction of Petri nets.

1 Introduction

The notion of equivalence is central to any theory of systems. Equivalences allow one to compare and reduce systems taking into account particular aspects of their behaviour. Petri nets became a popular formal model for design of concurrent and distributed systems. In recent years, a wide range of behavioural equivalences was proposed in the concurrency theory. The equivalencs can be classified depending of semantics of concurrency they impose. In *interleaving* semantics, a concurrent happening of actions is interpreted as their occurrence in any possible order. In *step* semantics, a concurrency of actions is a basic notion, but their causal dependencies are not respected. In *partial word* semantics, causal dependencies of actions are respected in part via partially ordered multisets (pomsets) of actions, and a pomset may be modelled by a less sequential one (i.e. having less strict partial order). In *pomset* semantics, causal dependencies of actions are fully respected, and pomsets of actions should coincide to model each other. In *process* semantics, a structure of a process (causal) net is respected.

*The work is supported by Volkswagen Stiftung, grant I/70 564, INTAS-RFBR, grant 95-0378 and the Foundation for Promotion to Young Scientists of Siberian Division of the Russian Academy of Sciences

The following basic notions of behavioural equivalences were proposed:

- *Trace equivalences* (they respect only protocols of behaviour of systems): interleaving ($\equiv_i$) [8], step ($\equiv_s$) [8], partial word ($\equiv_{pw}$) [12], pomset ($\equiv_{pom}$) [8] and process ($\equiv_{pr}$) [10].

- *Usual bisimulation equivalences* (they respect branching structure of behaviour of systems): interleaving ($\underline{\leftrightarrow}_i$) [8], step ($\underline{\leftrightarrow}_s$) [8], partial word ($\underline{\leftrightarrow}_{pw}$) [12], pomset ($\underline{\leftrightarrow}_{pom}$) [8] and process ($\underline{\leftrightarrow}_{pr}$) [3].

- *ST-bisimulation equivalences* (they respect the duration or maximality of events in behaviour of systems): interleaving ($\underline{\leftrightarrow}_{iST}$) [7], partial word ($\underline{\leftrightarrow}_{pwST}$) [12], pomset ($\underline{\leftrightarrow}_{pomST}$) [12] and process ($\underline{\leftrightarrow}_{prST}$) [10].

- *History preserving bisimulation equivalences* (they respect the "history" of behaviour of systems): pomset ($\underline{\leftrightarrow}_{pomh}$) [12] and process ($\underline{\leftrightarrow}_{prh}$) [10].

- *Conflict preserving equivalences* (they completely respect conflicts of events in systems): multi event structure ($\equiv_{mes}$) [10] and occurrence ($\equiv_{occ}$) [7].

- *Isomorphism* ($\simeq$) (i.e. coincidence of systems up to renaming of their components).

Another important group of equivalences are back-forth bisimulation ones which are based on the idea that a bisimulation relation should not only require systems to simulate each other behaviour in the forward direction but also when going back in the history. By now, the set of all possible back-forth equivalence notions was proposed in interleaving, step, partial word and pomset semantics. Most of them coincide with basic or with other back-forth relations. The following new notions were obtained: step back step forth ($\underline{\leftrightarrow}_{sbsf}$) [6], step back partial word forth ($\underline{\leftrightarrow}_{sbpwf}$) [9] and step back pomset forth ($\underline{\leftrightarrow}_{sbpomf}$) [9] bisimulation equivalences. In [11] we supplemented them by several new relations in process semantics: step back process forth ($\underline{\leftrightarrow}_{sbprf}$) and pomset back process forth ($\underline{\leftrightarrow}_{pombprf}$) bisimulation equivalences.

The third important group of equivalences are place bisimulation ones introduced in [1]. They are relations between places (instead of markings or processes). The relation on markings is obtained using the "lifting" of relation on places. The main application of place bisimulation equivalences is an effective global behaviour preserving reduction technique for Petri nets based on them. In [1], interleaving place bisimulation equivalence ($\sim_i$) was proposed. In this paper, strict interleaving place bisimulation equivalence ($\approx_i$) was defined also, by imposing the additional requirement stating that corresponding transitions of nets must be related by the bisimulation. In [3, 4], step ($\sim_s$), partial word ($\sim_{pw}$), pomset ($\sim_{pom}$), process ($\sim_{pr}$) place bisimulation equivalences and their strict analogues ($\approx_s$, $\approx_{pw}$, $\approx_{pom}$, $\approx_{pr}$) were proposed. The coincidence of $\sim_i$, $\sim_s$ and $\sim_{pw}$ was established. It was shown that all strict bisimulation equivalences coincide with $\sim_{pr}$. Thus, only three different equivalences remain: $\sim_i$, $\sim_{pom}$ and $\sim_{pr}$. In addition, in these papers the polynomial algorithm of a net reduction modulo $\sim_i$ and $\sim_{pr}$ was proposed.

To choose appropriate behavioural viewpoint on systems to be modelled, it is important to have a complete set of equivalence notions in all semantics and understand their interrelations. Treating equivalences for preservation by refinements allows one to decide which of them may be used for top-down design. In this paper, we obtain a number of results on solution these problems for place bisimulation equivalences.

The first result is a diagram of interrelations of place equivalences with basic and back-forth behavioural notions from [10, 11]. We prove that $\sim_{pr}$ implies $\underline{\leftrightarrow}_{prh}$ and answer the question from [1]: it is no sense to define history preserving place bisimulation equivalence. Another consequence is: the algorithm of a net reduction from [3, 4], based on $\sim_{pr}$, preserves "histories" of the behaviour of the initial net.

The second result is concerned a notion of transition refinement. In [5], SM-refinement operator for Petri nets was proposed, which "replaces" their transitions by SM-nets, a subclass of state machine nets. We treat all the considered equivalence notions for preservation by SM-refinements and establish that $\sim_{pr}$ is the only place bisimulation equivalence which is preserved by SM-refinements. Thus, this equivalence may be used for the compositional reduction of nets.

2 Basic definitions

In this section, we present some basic definitions used further.

2.1 Nets

Let $Act = \{a, b, \ldots\}$ be a set of *action names*.

Definition 2.1 *A* labelled net *is a quadruple* $N = \langle P_N, T_N, F_N, l_N \rangle$*, where:*

- $P_N = \{p, q, \ldots\}$ *is a set of* places*;*
- $T_N = \{t, u, \ldots\}$ *is a set of* transitions*;*
- $F_N : (P_N \times T_N) \cup (T_N \times P_N) \to \mathbf{N}$ *is the* flow relation *with weights (*$\mathbf{N}$ *denotes a set of natural numbers);*
- $l_N : T_N \to Act$ *is a* labelling *of transitions with action names.*

Given labelled nets N and N' A mapping $\beta : P_N \cup T_N \to P_{N'} \cup T_{N'}$ is an *isomorphism* between N and N', denoted by $\beta : N \simeq N'$, if β is a bijective renaming of places and transitions of N s.t. the nets N and N' coincide up to it. Two labelled nets N and N' are *isomorphic*, denoted by $N \simeq N'$, if $\exists \beta : N \simeq N'$.

Given a labelled net N and some transition $t \in T_N$, the *precondition* and *postcondition* of t, denoted by ${}^\bullet t$ and $t^\bullet$ respectively, are the multisets defined in such a way: $({}^\bullet t)(p) = F_N(p, t)$ and $(t^\bullet)(p) = F_N(t, p)$. Analogous definitions are introduced for places: $({}^\bullet p)(t) =$

$F_N(t,p)$ and $(p^\bullet)(t) = F_N(p,t)$. Let ${}^\circ N = \{p \in P_N \mid {}^\bullet p = \emptyset\}$ is the set of *input* places of N and $N^\circ = \{p \in P_N \mid p^\bullet = \emptyset\}$ is the set of *output* places of N.

A labelled net N is *acyclic*, if there exist no transitions $t_0, \ldots, t_n \in T_N$ s.t. $t_{i-1}^\bullet \cap {}^\bullet t_i \neq \emptyset$ $(1 \leq i \leq n)$ and $t_0 = t_n$. A labelled net N is *ordinary*, if $\forall p \in P_N$ ${}^\bullet p$ and $p^\bullet$ are proper sets (not multisets).

Let $N = \langle P_N, T_N, F_N, l_N \rangle$ be an acyclic ordinary labelled net and $x, y \in P_N \cup T_N$. Let us introduce the following notions.

- $x \prec_N y \;\Leftrightarrow\; x F_N^+ y$, where F_N^+ is a transitive closure of F_N (*the strict causal dependence* relation);
- $\downarrow_N x = \{y \in P_N \cup T_N \mid y \prec_N x\}$ (the set of *strict predecessors* of x);

A set $T \subseteq T_N$ is *left-closed* in N, if $\forall t \in T$ $(\downarrow_N t) \cap T_N \subseteq T$.

We denote the *set of all finite multisets* over a set X by $\mathcal{M}(X)$. A *marking* of a labelled net N is a multiset $M \in \mathcal{M}(P_N)$.

Definition 2.2 *A* (marked) net *is a tuple* $N = \langle P_N, T_N, F_N, l_N, M_N \rangle$, *where* $\langle P_N, T_N, F_N, l_N \rangle$ *is a labelled net and* $M_N \in \mathcal{M}(P_N)$ *is the* initial *marking.*

Let $M \in \mathcal{M}(P_N)$ be a marking of a net N. A transition $t \in T_N$ is *firable* in M, if ${}^\bullet t \subseteq M$. If t is firable in M, its firing yields a new marking $\widetilde{M} = M \Leftrightarrow {}^\bullet t + t^\bullet$, denoted by $M \stackrel{t}{\rightarrow} \widetilde{M}$.

2.2 Partially ordered sets

Definition 2.3 *A* labelled partially ordered set (lposet) *is a triple* $\rho = \langle X, \prec, l \rangle$, *where:*

- $X = \{x, y, \ldots\}$ *is some set;*
- $\prec \subseteq X \times X$ *is a strict partial order (irreflexive transitive relation) over* X*;*
- $l : X \rightarrow Act$ *is a* labelling *function.*

Let $\rho = \langle X, \prec, l \rangle$ and $\rho' = \langle X', \prec', l' \rangle$ be lposets.

A mapping $\beta : X \rightarrow X'$ is a *homomorphism* between ρ and ρ', denoted by $\beta : \rho \sqsubseteq \rho'$, if it is a bijection and $\forall x, y \in X$ $x \prec y \;\Rightarrow\; \beta(x) \prec' \beta(y)$, $\forall x \in X$ $l(x) = l'(\beta(x))$. We write $\rho \sqsubseteq \rho'$, if $\exists \beta : \rho \sqsubseteq \rho'$.

A mapping $\beta : X \rightarrow X'$ is an *isomorphism* between ρ and ρ', denoted by $\beta : \rho \simeq \rho'$, if $\beta : \rho \sqsubseteq \rho'$ and $\beta^{-1} : \rho' \sqsubseteq \rho$. Two lposets ρ and ρ' are *isomorphic*, denoted by $\rho \simeq \rho'$, if $\exists \beta : \rho \simeq \rho'$.

Definition 2.4 Partially ordered multiset (pomset) *is an isomorphism class of lposets.*

2.3 Processes

Definition 2.5 *A* causal net *is an acyclic ordinary labelled net* $C = \langle P_C, T_C, F_C, l_C \rangle$, *s.t.:*

1. $\forall r \in P_C \ |{}^\bullet r| \leq 1$ *and* $|r^\bullet| \leq 1$, *i.e. places are unbranched;*
2. $\forall x \in P_C \cup T_C \mid \downarrow_C x| < \infty$, *i.e. a set of causes is finite.*

Let us note that on the basis of any causal net C one can define lposet $\rho_C = \langle T_C, \prec_N \cap (T_C \times T_C), l_C \rangle$.

The fundamental property of causal nets is [3]: if C is a causal net, then there exists a sequence of transition firings (a *full execution* of C) s.t. ${}^\circ C = L_0 \stackrel{v_1}{\rightarrow} \cdots \stackrel{v_n}{\rightarrow} L_n = C^\circ$ s.t. $L_i \subseteq P_C$ $(0 \leq i \leq n)$, $P_C = \cup_{i=0}^n L_i$ and $T_C = \{v_1, \ldots, v_n\}$.

Definition 2.6 *Given a net* N *and a causal net* C. *A mapping* $\varphi : P_C \cup T_C \rightarrow P_N \cup T_N$ *is an* embedding *of* C *into* N, *denoted by* $\varphi : C \rightarrow N$, *if:*

1. $\varphi(P_C) \in \mathcal{M}(P_N)$ *and* $\varphi(T_C) \in \mathcal{M}(T_N)$, *i.e. sorts are preserved;*
2. $\forall v \in T_C \ {}^\bullet\varphi(v) = \varphi({}^\bullet v)$ *and* $\varphi(v)^\bullet = \varphi(v^\bullet)$, *i.e. flow relation is respected;*
3. $\forall v \in T_C \ l_C(v) = l_N(\varphi(v))$, *i.e. labelling is preserved.*

Since embeddings respect the flow relation, if ${}^\circ C \stackrel{v_1}{\rightarrow} \cdots \stackrel{v_n}{\rightarrow} C^\circ$ is a full execution of C, then $M = \varphi({}^\circ C) \stackrel{\varphi(v_1)}{\Leftrightarrow} \cdots \stackrel{\varphi(v_n)}{\Leftrightarrow} \varphi(C^\circ) = \widetilde{M}$ is a sequence of transition firings in N.

Definition 2.7 *A* firable in marking M process *of a net* N *is a pair* $\pi = (C, \varphi)$, *where* C *is a causal net and* $\varphi : C \rightarrow N$ *is an embedding s.t.* $M = \varphi({}^\circ C)$. *A firable in* M_N *process is a* process *of* N.

We write $\Pi(N, M)$ for the *set of all firable* in marking M processes of a net N and $\Pi(N)$ for the *set of all processes* of a net N. The *initial* process of a net N is $\pi_N = (C_N, \varphi_N) \in \Pi(N)$, s.t. $T_{C_N} = \emptyset$. If $\pi \in \Pi(N, M)$, then firing of this process transforms a marking M into $\widetilde{M} = \varphi(C^\circ)$, denoted by $M \stackrel{\pi}{\rightarrow} \widetilde{M}$.

Let $\pi = (C, \varphi)$, $\tilde{\pi} = (\widetilde{C}, \tilde{\varphi}) \in \Pi(N)$, $\hat{\pi} = (\widehat{C}, \hat{\varphi}) \in \Pi(N, \varphi(C^\circ))$. A process $\tilde{\pi}$ is an *extension* of π *by process* $\hat{\pi}$, denoted by $\pi \stackrel{\hat{\pi}}{\rightarrow} \tilde{\pi}$, if $T_C \subseteq T_{\widetilde{C}}$ is a left-closed set in $\widetilde{C}$ and $T_{\widehat{C}} = T_{\widetilde{C}} \setminus T_C$. We write $\pi \rightarrow \tilde{\pi}$, if $\exists \hat{\pi} \ \pi \stackrel{\hat{\pi}}{\rightarrow} \tilde{\pi}$. A process $\tilde{\pi}$ is an extension of π *by one transition*, denoted by $\pi \stackrel{v}{\rightarrow} \tilde{\pi}$, if $\pi \stackrel{\hat{\pi}}{\rightarrow} \tilde{\pi}$ and $T_{\widehat{C}} = \{v\}$.

3 Place bisimulation equivalences

In this section, place bisimulation equivalences are introduced. Let us recall the definition of usual bisimulation equivalences.

Definition 3.1 *Let N and N' be some nets. A relation $\mathcal{R} \subseteq \mathcal{M}(N) \times \mathcal{M}(N')$ is a* $\star$-bisimulation *between N and N', $\star \in$ {interleaving, step, partial word, pomset, process}, denoted by $\mathcal{R} : N \underline{\leftrightarrow}_\star N'$, $\star \in \{i, s, pw, pom, pr\}$, if:*

1. $(M_N, M_{N'}) \in \mathcal{R}$.
2. $(M, M') \in \mathcal{R}$, $M \stackrel{\hat{\pi}}{\rightarrow} \widetilde{M}$,
   (a) $|T_{\widehat{C}}| = 1$, *if* $\star = i$;
   (b) $\prec_{\widehat{C}} = \emptyset$, *if* $\star = s$;

   $\Rightarrow \exists \widetilde{M}' :\ M' \stackrel{\hat{\pi}'}{\rightarrow} \widetilde{M}'$, $(\widetilde{M}, \widetilde{M}') \in \mathcal{R}$ *and*

   (a) $\rho_{\widehat{C}'} \sqsubseteq \rho_{\widehat{C}}$, *if* $\star = pw$;
   (b) $\rho_{\widehat{C}} \simeq \rho_{\widehat{C}'}$, *if* $\star \in \{i, s, pom\}$;
   (c) $\widehat{C} \simeq \widehat{C}'$, *if* $\star = pr$.
3. *As item 2, but the roles of N and N' are reversed.*

Two nets N and N' are $\star$-bisimulation equivalent, $\star \in$ {interleaving, step, partial word, pomset, process}, *denoted by $N \underline{\leftrightarrow}_\star N'$, if $\exists \mathcal{R} : N \underline{\leftrightarrow}_\star N'$, $\star \in \{i, s, pw, pom, pr\}$.*

Place bisimulations are relations between places instead of markings. A relation on markings is obtained with use of the "lifting" of a bisimulation relation on places.

Let for nets N and N' $\mathcal{R} \subseteq P_N \times P_{N'}$ be a relation between their places. The *lifting* of $\mathcal{R}$ is a relation $\overline{\mathcal{R}} \subseteq \mathcal{M}(P_N) \times \mathcal{M}(P_{N'})$, defined as follows: $(M, M') \in \overline{\mathcal{R}} \Leftrightarrow \exists \{(p_1, p_1'), \ldots, (p_n, p_n')\} \in \mathcal{M}(\mathcal{R}) :\ M = \{p_1, \ldots p_n\},\ M' = \{p_1', \ldots p_n'\}$.

Definition 3.2 *Let N and N' be some nets. A relation $\mathcal{R} \subseteq P_N \times P_{N'}$ is a* $\star$-place bisimulation *between N and N', $\star \in$*{interleaving, step, partial word, pomset, process}, *denoted by $\mathcal{R} : N \sim_\star N'$, if $\overline{\mathcal{R}} : N \underline{\leftrightarrow}_\star N'$, $\star \in \{i, s, pw, pom, pr\}$.*

Two nets N and N' are $\star$-place bisimulation equivalent, $\star \in$*{interleaving, step, partial word, pomset, process}, denoted by $N \sim_\star N'$, if $\exists \mathcal{R} : N \sim_\star N'$, $\star \in \{i, s, pw, pom, pr\}$.*

Strict place bisimulation equivalences are defined using the additional requirement stating that corresponding transitions of nets must be (as well as makings) related by $\overline{\mathcal{R}}$. This relation is defined on transitions as follows.

Let for some nets N and N' $t \in T_N$, $t' \in T_{N'}$. Then $(t, t') \in \overline{\mathcal{R}} \Leftrightarrow ({}^\bullet t, {}^\bullet t') \in \overline{\mathcal{R}}) \wedge ((t^\bullet, t'^\bullet) \in \overline{\mathcal{R}}) \wedge (l_N(t) = l_{N'}(t'))$.

Definition 3.3 *Let N and N' be some nets. A relation $\mathcal{R} \subseteq P_N \times P_{N'}$ is a* strict $\star$-place bisimulation *between N and N', $\star \in$*{interleaving, step, partial word, pomset, process}, *denoted by $\mathcal{R} : N \approx_\star N'$, $\star \in \{i, s, pw, pom, pr\}$, if:*

1. $\overline{\mathcal{R}} : N \underline{\leftrightarrow}_{\star} N'$.

2. *In the definition of* $\star$*-bisimulation in item 2 (and in item 3 symmetrically) the new requirement is added:* $\forall v \in T_{\widehat{C}}\ (\hat{\varphi}(v), \hat{\varphi}'(\beta(v))) \in \overline{\mathcal{R}}$, *where:*

   (a) $\beta : \rho_{\widehat{C}'} \sqsubseteq \rho_{\widehat{C}}$, *if* $\star = pw$;

   (b) $\beta : \rho_{\widehat{C}} \simeq \rho_{\widehat{C}'}$, *if* $\star \in \{i, s, pom\}$;

   (c) $\beta : \widehat{C} \simeq \widehat{C}'$, *if* $\star = pr$.

Two nets N *and* N' *are* strict $\star$-place bisimulation equivalent, $\star \in$\{interleaving, step, partial word, pomset, process\}, *denoted by* $N \approx_{\star} N'$, *if* $\exists \mathcal{R} : N \approx_{\star} N'$, $\star \in \{i, s, pw, pom, pr\}$.

An important property of place bisimulations is *additivity*. Let for nets N and N' $\mathcal{R} : N \sim_{\star} N'$. Then $(M_1, M_1') \in \overline{\mathcal{R}}$ and $(M_2, M_2') \in \overline{\mathcal{R}}$ implies $((M_1 + M_2), (M_1' + M_2')) \in \overline{\mathcal{R}}$. In particular, if we put n tokens into each of the places $p \in P_N$ and $p' \in P_{N'}$ s.t. $(p, p') \in \mathcal{R}$, then the nets obtained as a result of such a changing of the initial markings, must be also place bisimulation equivalent.

The following proposition establishes a coincidence of most place bisimulation equivalences.

Proposition 3.1 *[3, 4] For nets* N *and* N'*:*

1. $N \sim_i N' \ \Leftrightarrow \ N \sim_{pw} N'$;

2. $N \sim_{pr} N' \ \Leftrightarrow \ N \approx_i N' \ \Leftrightarrow \ N \approx_{pr} N'$.

4 Interrelations of the equivalences

In this section, place bisimulation equivalences are compared with basic equivalences and back-forth bisimulation equivalences. First, recall the definition of history preserving bisimulation equivalences.

Definition 4.1 *Let* N *and* N' *be some nets. A relation* $\mathcal{R} \subseteq \Pi(N) \times \Pi(N') \times \mathcal{B}$, *where* $\mathcal{B} = \{\beta \mid \beta : T_C \to T_{C'},\ \pi = (C, \varphi) \in \Pi(N),\ \pi' = (C', \varphi') \in \Pi(N')\}$, *is a* $\star$-history preserving bisimulation *between* N *and* N', $\star \in$\{pomset, process\}, *denoted by* $N \underline{\leftrightarrow}_{\star h} N'$, $\star \in \{pom, pr\}$, *if:*

1. $(\pi_N, \pi_{N'}, \emptyset) \in \mathcal{R}$.

2. $(\pi, \pi', \beta) \in \mathcal{R} \ \Rightarrow$

   (a) $\beta : \rho_C \simeq \rho_{C'}$, *if* $\star \in \{pom, pr\}$;

   (b) $C \simeq C'$, *if* $\star = pr$.

3. $(\pi, \pi', \beta) \in \mathcal{R},\ \pi \to \tilde{\pi} \ \Rightarrow \ \exists \tilde{\beta},\ \tilde{\pi}' : \pi' \to \tilde{\pi}',\ \tilde{\beta}|_{T_C} = \beta,\ (\tilde{\pi}, \tilde{\pi}', \tilde{\beta}) \in \mathcal{R}$.

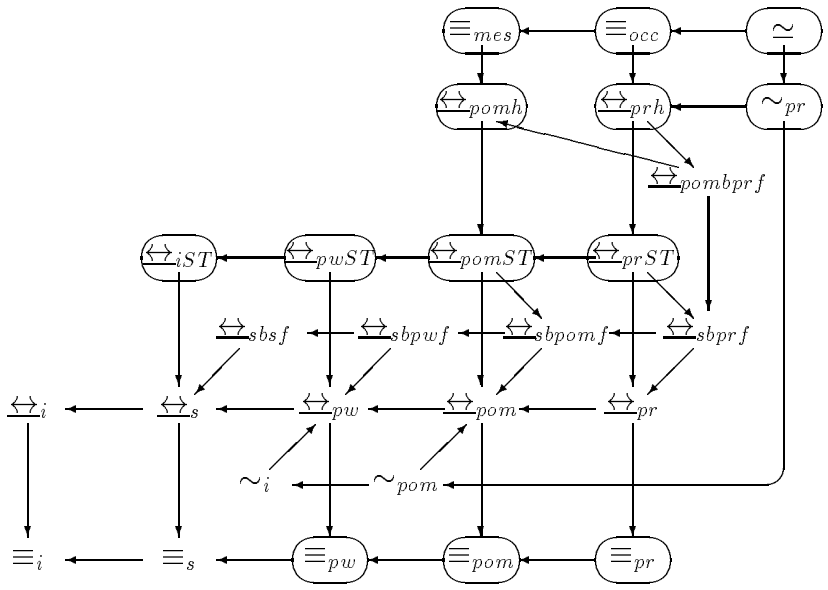


Figure 1: Interrelations of the equivalences and their preservation by SM-refinements

4. As item 3 but the roles of N and N' are reversed.

Two nets N and N' are $\star$-history preserving bisimulation *equivalent,* $\star \in$ {pomset, process}, *denoted by $N \underline{\leftrightarrow}_{\star h} N'$, if $\exists \mathcal{R} : N \underline{\leftrightarrow}_{\star h} N'$, $\star \in \{pom, pr\}$.*

Let us note that in this definition one can use extentions of processes by one transition only. Now we are able to prove the proposition about interrelations of place and history preserving equivalences.

Proposition 4.1 *For nets N and N': $N \sim_{pr} N' \Rightarrow N \underline{\leftrightarrow}_{prh} N'$.*

Proof. See Appendix A. □

Below, the symbol '_' will denote "nothing", and the signs of equivalences subscribed by it are considered as that of without any subscribtion. The following theorem collect all the results obtained here and in [11], and clarify interrelations of all the equivalences.

Theorem 4.1 *Let $\leftrightarrow, \twoheadleftarrow \in \{\equiv, \underline{\leftrightarrow}, \sim, \simeq\}$, $\star, \star\star \in \{\_, i, s, pw, pom, pr, iST, pwST, pomST, prST, pomh, prh, mes, occ, sbsf, sbpwf, sbpomf, sbprf, pombprf\}$. For nets N and N': $N \leftrightarrow_\star N' \Rightarrow N \twoheadleftarrow_{\star\star} N'$ iff in the graph in Figure 1 there exists a directed path from $\leftrightarrow_\star$ to $\twoheadleftarrow_{\star\star}$.*

Proof. ($\Leftarrow$) By Theorem 12 from [11] and the following substantiations.

- The implications $\sim_\star \rightarrow \underline{\leftrightarrow}_\star$, $\star \in \{i, pom, pr\}$ are valid by the definitions.
- The implication $\sim_{pr} \rightarrow \underline{\leftrightarrow}_{prh}$ is valid by Proposition 3.2.

- The implication $\sim_{pom} \rightarrow \sim_i$ is valid by the definitions.
- The implication $\sim_{pr} \rightarrow \sim_{pom}$ is valid since lposets of isomorphic nets are also isomorphic.
- The implication $\simeq \rightarrow \sim_{pr}$ is obvious.

($\Rightarrow$) By Theorem 12 from [11] and the following examples (dashed lines in Figure 2 connect bisimilar places).

- In Figure 2(a), $N \sim_i N'$, but $N \not\equiv_{pom} N'$, since only in the net N' action b can depend on a.
- In Figure 2(b), $N \sim_{pom} N'$, but $N \not\equiv_{pr} N'$, since only in the net N' the transition with label a has two input (and two output) places.
- In Figure 2(c), $N \equiv_{occ} N'$, but $N \not\sim_i N'$, since any place bisimulation must relate input places of the nets N and N'. But after putting one additional token into each of these places only in N' the action c can happen.
- In Figure 2(b), $N \sim_{pom} N'$, but $N \underline{\not\leftrightarrow}_{iST} N'$, since only in the net N' action a can start so that no b can begin working until finishing of a.
- In Figure 2(d), $N \sim_{pr} N'$, but $N \not\equiv_{mes} N'$, since only the net N' has two conflict actions a.
- In Figure 2(b), $N \sim_{pom} N'$, but $N \underline{\not\leftrightarrow}_{sbsf} N'$, since only in the net N' action a can happen so that b must depend on a. □

In this section, we obtained a number of important results. Before, place bisimulation equivalences have been compared with usual bisimulation ones only. Here, we clarified their interrelations with all the basic and back-forth ones. We proved that $\sim_{pom}$ does not imply neiter ST- nor back-forth bisimulation equivalences. The situation is quite different for $\sim_{pr}$. It appears to be strict enough to imply history preserving bisimulation equivalences. This interesting result may be used in reduction of nets modulo $\sim_{pr}$ [3, 4]. Now, we can guarantee that the reduced net has the same histories of the behaviour as the initial one.

5 Preservation of the equivalences by refinements

In this section, we treat the considered equivalence notions for preservation by transition refinements. We use SM-refinement, i.e. refinement by a special subclass of state-machine nets introduced in [5].

Definition 5.1 *An* SM-net *is a net* $D = \langle P_D, T_D, F_D, l_D, M_D \rangle$ *s.t.:*

1. $\forall t \in T_D \ |{}^\bullet t| = |t^\bullet| = 1$, *i.e. each transition has exactly one input and one output place;*

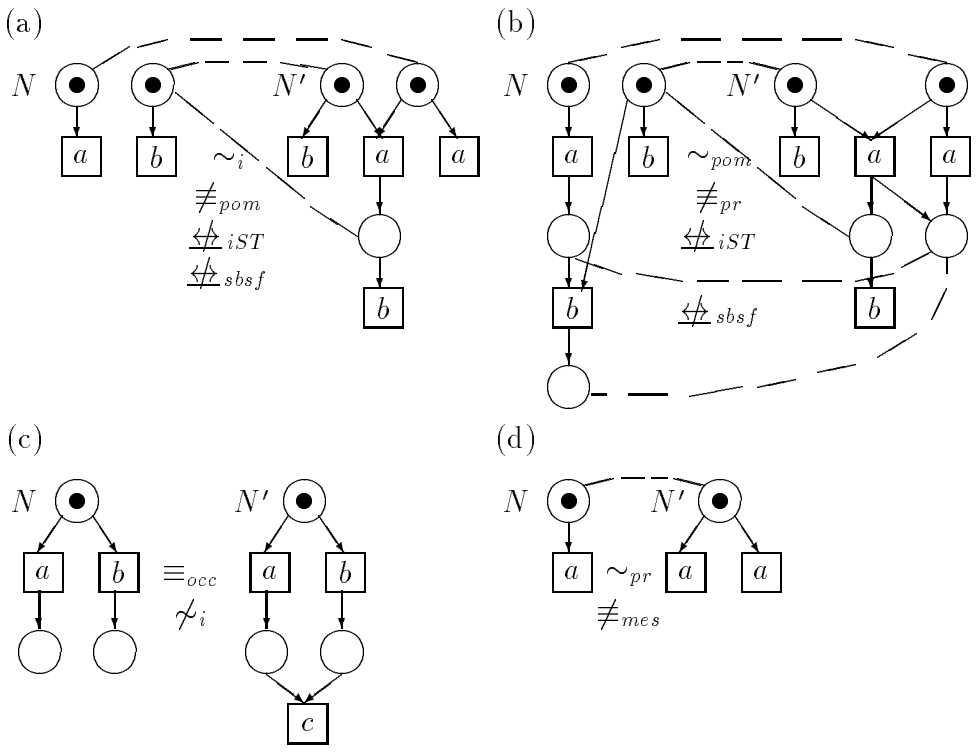


Figure 2: Examples of place bisimulation equivalences

2. $\exists p_{in},\ p_{out} \in P_D$ *s.t.* $p_{in} \neq p_{out}$ *and* ${}^\circ D = \{p_{in}\},\ D^\circ = \{p_{out}\}$, *i.e. it is an unique input and an unique output place.*

3. $M_D = \{p_{in}\}$, *i.e. at the beginning there is an unique token in* p_{in}.

Definition 5.2 *Let* $N = \langle P_N, T_N, F_N, l_N, M_N\rangle$ *be some net,* $a \in l_N(T_N)$ *and* $D = \langle P_D, T_D, F_D, l_D, M_D\rangle$ *be SM-net. An* SM-refinement, *denoted by* $ref(N, a, D)$, *is a net* $\overline{N} = \langle P_{\overline{N}}, T_{\overline{N}}, F_{\overline{N}}, l_{\overline{N}}, M_{\overline{N}}\rangle$, *where:*

- $P_{\overline{N}} = P_N \cup \{\langle p, u\rangle \mid p \in P_D \setminus \{p_{in}, p_{out}\},\ u \in l_N^{-1}(a)\}$;
- $T_{\overline{N}} = (T_N \setminus l_N^{-1}(a)) \cup \{\langle t, u\rangle \mid t \in T_D,\ u \in l_N^{-1}(a)\}$;
- $F_{\overline{N}}(\bar{x}, \bar{y}) = \begin{cases} F_N(\bar{x}, \bar{y}), & \bar{x}, \bar{y} \in P_N \cup (T_N \setminus l_N^{-1}(a)); \\ F_D(x, y), & \bar{x} = \langle x, u\rangle,\ \bar{y} = \langle y, u\rangle,\ u \in l_N^{-1}(a); \\ F_N(\bar{x}, u), & \bar{y} = \langle y, u\rangle,\ \bar{x} \in {}^\bullet u,\ u \in l_N^{-1}(a),\ y \in p_{in}^\bullet; \\ F_N(u, \bar{y}), & \bar{x} = \langle x, u\rangle,\ \bar{y} \in {}^\bullet u,\ u \in l_N^{-1}(a),\ x \in {}^\bullet p_{out}; \\ 0, & otherwise; \end{cases}$
- $l_{\overline{N}}(\bar{u}) = \begin{cases} l_N(\bar{u}), & \bar{u} \in T_N \setminus l_N^{-1}(a); \\ l_D(t), & \bar{u} = \langle t, u\rangle,\ t \in T_D,\ u \in l_N^{-1}(a); \end{cases}$

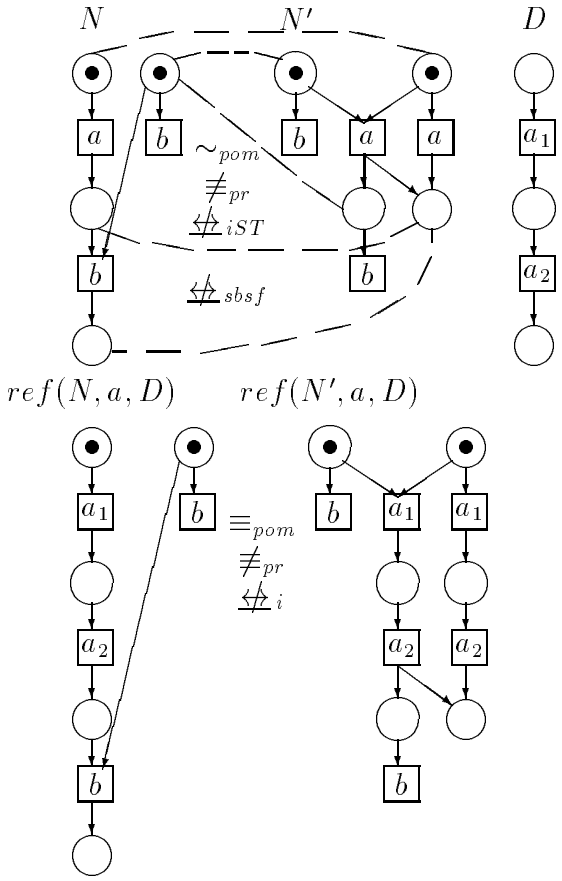


Figure 3: The equivalences between $\underline{\leftrightarrow}_i$ and $\sim_{pom}$ are not preserved by SM-refinements

- $M_{\overline{N}}(p) = \begin{cases} M_N(p), & p \in P_N; \\ 0, & otherwise. \end{cases}$

An equivalence is *preserved by refinements*, if equivalent nets remain equivalent after applying any refinement operator to them accordingly. The following proposition demonstrates that some place equivalences are not preserved by SM-refinements.

Proposition 5.1 *The equivalences $\sim_i$ and $\sim_{pom}$ are not preserved by SM-refinements.*

Proof. In Figure 3, $N \sim_{pom} N'$, but $ref(N,a,D) \not\underline{\leftrightarrow}_i ref(N',a,D)$, since only in the net $ref(N',a,D)$ after action a_1 action b cannot happen. Consequently, equivalences between $\underline{\leftrightarrow}_i$ and $\sim_{pom}$ are not preserved by SM-refinements. □

The following proposition proves that $\sim_{pr}$ is preserved by refinements.

Proposition 5.2 *For nets N and N' s.t. $a \in l_N(T_N) \cap l_{N'}(T_{N'})$ and SM-net D : $N \sim_{pr} N' \Rightarrow ref(N,a,D) \sim_{pr} ref(N',a,D)$.*

Proof. See Appendix B. □

Now we can add the results obtained to that of from [11] and present the following theorem.

Theorem 5.1 *Let* $\leftrightarrow \in \{\equiv, \underline{\leftrightarrow}, \sim, \simeq\}$ *and* $\star \in \{\_, i, s, pw, pom, pr, iST, pwST, pomST, prST, pomh, prh, mes, occ, sbsf, sbpwf, sbpomf, sbprf, pombprf\}$. *For nets* N *and* N' *s.t.* $a \in l_N(T_N) \cap l_{N'}(T_{N'})$ *and SM-net* D : $N \leftrightarrow_\star N' \Rightarrow ref(N, a, D) \leftrightarrow_\star ref(N', a, D)$ *iff the equivalence* $\leftrightarrow_\star$ *is in oval in Figure 1.*

Proof. By Theorem 18 from [11] and Propositions 5.1 and 5.2. □

In this section, an important result has been established. From all the place bisimulation equivalences, only $\sim_{pr}$ is preserved by refinements. Thus, it can be used for the compositional refinement of Petri nets.

For example, let us consider a net modelling a concurrent system and the reduced (modulo some equivalence) version of this net. The initial and the reduced nets have similar behaviour. Thus, we can use the reduced net instead of the initial one as a model for the concurrent system. If we want to consider the system at lower abstraction level, we use a refinement operation which "replaces" several transitions of the nets to the subnets corresponding to some internal structure of the system's components. If the equivalence used for the reduction is not preserved by refinements, we cannot use the refined reduced net as a model anymore, since its behaviour can be different with that of the refined initial net.

Hence, the preservation of $\sim_{pr}$ by refinements is a powerful property, especially if to remember that this equivalence implies the history preserving one. Consequently, the histories of behaviour of the initial net coincide with that of the reduced net, and this property is valid at different abstraction levels.

6 Conclusion

In this paper, we examined a group of place bisimulation equivalences. We compared them with basic and back-forth ones. All the considered equivalences were treated for preservation by SM-refinements to establish which of them may be used for top-down design of concurrent systems. We proved that $\sim_{pr}$ implies $\underline{\leftrightarrow}_{prh}$ and it is preserved by refinements. Hence, it may be used for the compositional and history-preserving reduction of concurrent systems modelled by Petri nets.

Further research may consist in the investigation of analogues of the considered equivalences on Petri nets with τ-actions (τ-equivalences). τ-actions are used to abstract of internal, invisible to external observer behaviour of systems to be modelled. Let us note that a number of interleaving place τ-bisimulation equivalences was proposed in [4, 2]. For other semantics, the corresponding relations have not been defined, and it would be interesting to propose them and exam their interrelations. In future, we plan to define τ-analogues of all the equivalence relations considered in this paper and exam them following the same pattern.

References

[1] Autant C., Belmesk Z., Schnoebelen Ph. *Strong bisimularity on nets revisited. Extended abstract. LNCS* **506**, p. 295–312, June 1991.

[2] Autant C., Pfister W., Schnoebelen Ph. *Place bisimulations for the reduction of labelled Petri nets with silent moves. Proceedings of International Conference on Computing and Information*, 1994.

[3] Autant C., Schnoebelen Ph. *Place bisimulations in Petri nets. LNCS* **616**, p. 45–61, June 1992.

[4] Autant C. *Petri nets for the semantics and the implementation of parallel processes. Ph.D. thesis*, Institut National Polytechnique de Grenoble, May 1993 (in French).

[5] Best E., Devillers R., Kiehn A., Pomello L. *Concurrent bisimulations in Petri nets. Acta Informatica* **28**, p. 231–264, 1991.

[6] Cherief F. *Back and forth bisimulations on prime event structures. LNCS* **605**, p. 843–858, June 1992.

[7] van Glabbeek R.J., Vaandrager F.W. *Petri net models for algebraic theories of concurrency. LNCS* **259**, p. 224–242, 1987.

[8] Pomello L., Rozenberg G., Simone C. *A survey of equivalence notions for net based systems. LNCS* **609**, p. 410–472, 1992.

[9] Pinchinat S. *Bisimulations for the semantics of reactive systems. Ph.D. thesis*, Institut National Politechnique de Grenoble, January 1993 (in French).

[10] Tarasyuk I.V. *Petri net equivalences for design of concurrent systems.* Proceedings of 5^{th} Workshop on Concurrency, Specification and Programming - 96 (CSP'96), September 25–27, 1996, *Informatik-Bericht* **69**, p. 190–204, Institut für Informatik, Humboldt-Universität zu Berlin, Berlin, Germany, 1996.

[11] Tarasyuk I.V. *Back-forth equivalences for design of concurrent systems.* S. Adian, A. Nerode, eds., Proceedings of 4^{th} International Symposium on Logical Foundations of Computer Science - 97 (LFCS'97), *LNCS* **1234**, p. 374–384, Yaroslavl, 1997.

[12] Vogler W. *Modular construction and partial order semantics of Petri nets. LNCS* **625**, 252 p., 1992.

A Proof of Proposition 4.1.

By Proposition 3.1, $\exists \mathcal{R} : N \approx_{pr} N'$. Then $\overline{\mathcal{R}} : N \underline{\leftrightarrow}_{pr} N'$ and transitions of N and N' are related by $\overline{\mathcal{R}}$. Let us define a relation $\mathcal{S}$ as follows: $\mathcal{S} = \{(\pi, \pi', \beta) \mid \pi = (C, \varphi) \in \Pi(N),\ \pi' = (C, \varphi') \in \Pi(N'),\ \beta = id_{T_C},\ \forall r \in P_C\ (\varphi(r), \varphi'(r)) \in \mathcal{R},\ \forall v \in T_C\ (\varphi(v), \varphi'(v)) \in \overline{\mathcal{R}}\}$. Let us prove $\mathcal{S} : N \underline{\leftrightarrow}_{prh} N'$.

1. Obviously, $(\pi_N, \pi_{N'}, \emptyset) \in \mathcal{S}$.

2. By definition of $\mathcal{S}$, $(\pi, \pi', \beta) \in \mathcal{S} \ \Rightarrow\ \beta : \rho_C \simeq \rho_{C'}$ and $C \simeq C'$;

3. Let $(\pi, \pi', \beta) \in \mathcal{S},\ \pi = (C, \varphi),\ \pi' = (C, \varphi')$ and $\pi \xrightarrow{v} \tilde{\pi},\ \tilde{\pi} = (\widetilde{C}, \tilde{\varphi})$.

   Let us consider a transition firing $\tilde{\varphi}({}^\bullet v) \xrightarrow{\tilde{\varphi}(v)} \tilde{\varphi}(v^\bullet)$ in N. By definition of $\mathcal{S}$, $(\varphi({}^\bullet v), \varphi'({}^\bullet v)) \in \overline{\mathcal{R}}$. Since $\varphi({}^\bullet v) = \tilde{\varphi}({}^\bullet v)$, we have $(\tilde{\varphi}({}^\bullet v), \varphi'({}^\bullet v)) \in \overline{\mathcal{R}}$.

   Since $\mathcal{R} : N \approx_{pr} N'$, we have $\exists u', \widetilde{M}' :\ \varphi'({}^\bullet v) \xrightarrow{u'} \widetilde{M}',\ (\tilde{\varphi}(v), u') \in \overline{\mathcal{R}}$ and $(\tilde{\varphi}(v^\bullet), \widetilde{M}') \in \overline{\mathcal{R}}$.

   Let $v^\bullet = \{r_1, \ldots, r_n\},\ \widetilde{M}' = \{p'_1, \ldots, p'_n\},\ \forall i\ (1 \le i \le n)\ (\tilde{\varphi}(r_i), p'_i) \in \mathcal{R}$. Let us define a mapping $\tilde{\varphi}'$ as follows: $\tilde{\varphi}'|_{(P_C \cup T_C)} = \varphi',\ \tilde{\varphi}'(v) = u',\ \forall i\ (1 \le i \le n)\ \tilde{\varphi}'(r_i) = p'_i$.

   Since by definition of $\tilde{\varphi}'$ we have $u' = \tilde{\varphi}'(v),\ \widetilde{M}' = \tilde{\varphi}'(v^\bullet),\ \varphi'({}^\bullet v) = \tilde{\varphi}'({}^\bullet v)$, then $\tilde{\varphi}'({}^\bullet v) \xrightarrow{\tilde{\varphi}'(v)} \tilde{\varphi}'(v^\bullet)$ is a transition firing in N' and $(\tilde{\varphi}(v), \tilde{\varphi}'(v)) \in \overline{\mathcal{R}},\ (\tilde{\varphi}(v^\bullet), \tilde{\varphi}'(v^\bullet)) \in \overline{\mathcal{R}}$.

   Consequently, $\tilde{\varphi}({}^\bullet v) \Leftrightarrow {}^\bullet\tilde{\varphi}(v) = \tilde{\varphi}(v^\bullet) \Leftrightarrow \tilde{\varphi}(v)^\bullet$ and $\tilde{\varphi}'({}^\bullet v) \Leftrightarrow {}^\bullet\tilde{\varphi}'(v) = \tilde{\varphi}'(v^\bullet) \Leftrightarrow \tilde{\varphi}'(v)^\bullet$. Because of additivity of place bisimulations and since $\tilde{\varphi}$ is an embedding, we have $(\emptyset, \tilde{\varphi}'({}^\bullet v) \Leftrightarrow {}^\bullet\tilde{\varphi}'(v)) \in \overline{\mathcal{R}}$ and $(\emptyset, \tilde{\varphi}'(v^\bullet) \Leftrightarrow \tilde{\varphi}'(v)^\bullet) \in \overline{\mathcal{R}}$. Consequently, $\tilde{\varphi}'({}^\bullet v) = {}^\bullet\tilde{\varphi}'(v)$ and $\tilde{\varphi}'(v^\bullet) = \tilde{\varphi}'(v)^\bullet$. Therefore $\tilde{\varphi}'$ is an embedding and $\tilde{\pi}' = (\widetilde{C}, \tilde{\varphi}') \in \Pi(N')$. We have $\pi' \xrightarrow{v} \tilde{\pi}'$. Let us define $\tilde{\beta} = id_{T_{\widetilde{C}}}$. Then $(\tilde{\pi}, \tilde{\pi}', \tilde{\beta}) \in \mathcal{S}$.

4. As item 3, but the roles of N and N' are reversed. □

B Proof of Proposition 5.2.

Let $\overline{N} = ref(N, a, D)$, $\overline{N}' = ref(N', a, D)$ and $\mathcal{R} : N \sim_{pr} N'$. By Proposition 3.1, $\mathcal{R} : N \approx_i N'$. It is enough to prove $\overline{N} \approx_i \overline{N}'$. Let us define a relation $\mathcal{S}$ as follows: $\mathcal{S} = \mathcal{R} \cup \{(\langle p, u\rangle, \langle p, u'\rangle) \mid p \in P_D \setminus \{p_{in}, p_{out}\},\ (u, u') \in \overline{\mathcal{R}}\}$. Let us prove $\mathcal{S} : \overline{N} \approx_i \overline{N}'$.

1. $(M_{\overline{N}}, M_{\overline{N}'}) \in \mathcal{S}$, since $(M_N, M_{N'}) \in \mathcal{R}$.

2. Let $(M, M') \in \mathcal{S}$ and $M \xrightarrow{\bar{u}} \widetilde{M}$. Two cases are possible:

   (a) $\bar{u} = u \in T_N$;

   (b) $\bar{u} = \langle t, u\rangle,\ t \in T_D,\ u \in T_N,\ l_N(u) = a$.

Let us consider the case (b), since the case (a) is obvious. Let ${}^\bullet t = \{p\}$, $t^\bullet = \{q\}$. Then we have:

$${}^\bullet\langle t,u\rangle = \begin{cases} {}^\bullet u, & t \in p_{in}^\bullet; \\ \langle p,u\rangle, & \text{otherwise.} \end{cases} \quad \langle t,u\rangle^\bullet = \begin{cases} u^\bullet, & t \in {}^\bullet p_{out}; \\ \langle q,u\rangle, & \text{otherwise.} \end{cases}$$

Four cases are possible:

(a) $t \in p_{in}^\bullet \cap {}^\bullet p_{out}$;

(b) $t \in p_{in}^\bullet \setminus {}^\bullet p_{out}$;

(c) $t \in {}^\bullet p_{out} \setminus p_{in}^\bullet$;

(d) $t \notin p_{in}^\bullet \cup {}^\bullet p_{out}$.

Let us consider the case (d), since the cases (a)–(c) are simpler. We have ${}^\bullet\langle t,u\rangle = \langle p,u\rangle \in M$. Since $(M,M') \in \overline{\mathcal{S}}$, by definition of $\mathcal{S}$ we have: $\exists u' \in T_N :\ (u,u') \in \overline{\mathcal{R}}$ and $(\langle p,u\rangle, \langle p,u'\rangle) \in \mathcal{S}$, $\langle p,u'\rangle \in M'$. Since ${}^\bullet\langle t,u'\rangle = \langle p,u'\rangle$, then $({}^\bullet\langle t,u\rangle, {}^\bullet\langle t,u'\rangle) \in \overline{\mathcal{S}}$, ${}^\bullet\langle t,u'\rangle \in M'$.

Then $\exists \widetilde{M}' :\ M' \stackrel{\langle t,u'\rangle}{\to} \widetilde{M}'$. We have: $l_{\overline{N}}(\langle t,u\rangle) = l_D(t) = l_{\overline{N}'}(\langle t,u'\rangle)$. Since $\langle t,u\rangle^\bullet = \langle q,u\rangle$, by definition of $\mathcal{S}$ we have $(\langle q,u\rangle, \langle q,u'\rangle) \in \mathcal{S}$. Since $\langle t,u'\rangle^\bullet = \langle q,u'\rangle$, then $(\langle t,u\rangle^\bullet, \langle t,u'\rangle^\bullet) \in \overline{\mathcal{S}}$.

Hence, $(\langle t,u\rangle, \langle t,u'\rangle) \in \overline{\mathcal{S}}$ and $(\widetilde{M}, \widetilde{M}') \in \overline{\mathcal{S}}$.

3. As item 2, but the roles of $\overline{N}$ and $\overline{N}'$ are reversed. □

On the Semantics of Concurrency and Nondeterminism: Bisimulations and Temporal Logics*

Irina Virbitskaite
Institute of Informatics Systems
Siberian Division of the Russian Academy of Sciences
6, Lavrentieva ave.
630090, Novosibirsk, Russia
E-mail: virb@iis.nsk.su

Abstract

Event structures have come to play an important role in the formal study of the behaviour of distributed systems. The advantage of event structures is that they explicitly exhibit the interplay between concurrency and nondeterminism. This paper is contributed to develop a number of new bisimulations which are natural and nicely fit with the concept of event structures. We establish the closed relationships between the bisimulations, resulting in a lattice of implications. Some new logics with a CTL^* flavour, being interpreted over event structures, are further proposed to characterize the introduced bisimulations logically.

1 Introduction

Event structures have come to play an important role in the formal study of the behaviour of distributed systems. An event structure is a partially ordered set of events together with a symmetric conflict relation. The ordering relation models causality, whereas the conflict relation expresses alternative choices between events. Two events that are neither comparable nor in conflict, may occur concurrently. In this sense, event structures provide explicit and separate representations of causality, choice and concurrency.

Over the past several years, various equivalence notions have been defined on the domain of event structures. The best known behavioural equivalence is bisimulation. One of the

*This work is supported in part by the INTAS-RFBR (grant No 95-0378), the Volkswagen Foundation (grant No I/70 564), and and the Russian State Committee of High Education for Basic Research in Mathematics.

measures of success for a behavioural equivalence is its accompanying theory. And here bisimulation is particularly rich in results. However the standard definition of bisimulation can be applied only to systems whose operational behaviour is modelled by sequences of atomic actions and hence concurrency of actions is reduced to an arbitrary nondeterministic interleaving. Many attempts have been made to overcome the limits of this interleaving approach and to allow observer to discriminate systems via bisimulation also accordingly to the degree of concurrency they exploit in their computations. As a result, various equivalences based on modelling causal relations explicitly by partial orders have appeared in the literature. One such equivalence is history preserving bisimulation that was originally proposed by [18] for Petri nets and then adapted by [7] to event structures. Its many desirable properties have led to an in-depth study of history preserving bisimulation. For example, [20] established its decidability over finite safe Petri nets. Several other semantical characterizations exist (see [5], for example).

In parallel with the definition of behavioural equivalences, different attempts have been made towards defining modal and temporal logics that permit specifying specific properties of concurrent systems. Since logics naturally give rise to equivalence classes consisting of all those systems which satisfy the same formulas, often the logics known from the literature have been compared with behavioural equivalences for a better understanding and evaluation. In general, establishing a direct correspondence between logical and behavioural equivalences provides additional confidence in both approaches. A classical result here is the adequacy theorem of Hennessy and Milner stating that the logic HML, which when interpreted over labelled transition systems, is in full agreement with interleaving bisimulation. Following this direction, many other behavioural equivalences have been characterized through modal logics: see [3, 8, 10, 13, 19] among others.

There have been various motivations for this work. One has been given by the papers [4, 10, 15], where different back and forth forms of bisimulation have been defined and compared in the context of event structures. However, all these bisimulations capture intuition concerning causality and concurrency (implicitly), but not conflict between events in the structures. Attempting to get around this lack, we introduced a number of variants of interleaving and history preserving bisimulations which take into consideration all the relations between events, and therefore fit nicely with the concept of event structures. A next origin of this study was the logic CTL^* first proposed in [6] as a logic which included all other previously proposed temporal logics. Among many other applications, CTL^* like logics have been used as a benchmark for semantic equivalences. It was first shown that a variant of interleaving bisimulation coincides with the equivalence induced by CTL^* [3] and then that CTL^* without next operator is in full agreement with branching bisimulation [13]. These two results assume an interleaving setting. The paper [8] was a first welcome exception for giving CTL^* characterizations in an event structure setting. Further, [16] provided a logical characterization of history preserving bisimulation in terms of a path logic with a CTL^* flavour that uses pomsets observations. Finally, another origin for this work was the papers [9, 11], where different extensions of the CTL^* logic with past combinators have been defined, and the paper [12], where the logic L_1, having modalities

expressing concurrency and nondeterminism, has been put forward in the framework of event structures. While working further on enhancement of CTL^* expressivity, we looked for logics that could indeed express causality, concurrency and nondeterminism between events in the structures and would be characteristic for the introduced bisimulations.

The remainder of the paper is organised as follows. The next section defines the basic framework, labelled prime event structures, and related notions. In Section 3, we first suggest a number of variants of interleaving and history preserving bisimulations, which respect all the relations between event occurrences in the structures. Further, a lattice of the interrelations between the considered equivalences is constructed. Section 4 defines a number of extensions of CTL^* which are proven to be characteristic for the bisimulations. Finally, some concluding remarks are made in Section 5.

2 Event Structures

Event structures are well-known "truly concurrent" models of distributed systems. They have been introduced by Nielsen, Plotkin and Winskel. See [14, 21] for motivations and a complete technical exposition.

The main idea behind event structures is to view distributed computations as action occurrences, called events, together with a notion of causality dependency between events (which reasonably characterized via a partial order). Moreover, in order to model nondeterminism, there is a notion of conflicting (mutually incompatible) events. A labelling function records which action an event corresponds to.

Definition 2.1. A *(labelled) event structure* over an alphabet Act is a 4-tuple $\mathcal{E} = (E, <, \#, l)$, where

- E is a countable set of events;
- $< \subseteq E \times E$ is an irreflexive partial order (the *causality relation*), satisfying the *principle of finite causes*:
  $$\forall e \in E \circ \{d \in E \mid d < e\} \text{ is finite;}$$
- $\# \subseteq E \times E$ is a symmetric and irreflexive relation (the *conflict relation*) satisfying the *principle of conflict heredity*:
  $$\forall e_1, e_2, e_3 \in E \circ e_1 < e_2 \ \&\ e_1 \# e_3 \Rightarrow e_2 \# e_3;$$
- $l : E \to Act$ is a labelling function. ·

Through the paper, we assume a fixed set Act of action names (labels). The components of an event structure $\mathcal{E}$ are denoted by E_ε, $<_\varepsilon$, $\#_\varepsilon$ and l_ε. If clear from the context, the index $\mathcal{E}$ is omitted. For an event structure $\mathcal{E}$, we let: $id = \{(e, e) \mid e \in E\}$; $\leq = < \cup id$; $<^2 \subseteq <$ (transitivity); $\lessdot = < \setminus <^2$; $\smile = (E \times E) \setminus (\leq \cup \leq^{-1} \cup \#)$ (concurrency); $co = \smile \cup id$; $e \#_m d \stackrel{def}{\iff} e \# d \ \&\ \forall e', d' \in E \circ (e' \leq e \ \&\ d' \leq d \ \&\ e' \# d') \Rightarrow (e' = e \ \&\ d' = d)$ (minimal conflict).

In a graphic representation of an event structure, only minimal conflicts — not the inherited ones — are pictured. The $<$-relation is represented by arcs, omitting those derivable by transitivity. Following these conventions, a trivial example of an event structure is shown in Fig. 1, where $E = \{e_1, e_2, e_3, e_4\}$, $< = \{(e_1, e_3), (e_1, e_4), (e_2, e_3), (e_2, e_4)\}$, $\# = \{(e_3, e_4), (e_4, e_3)\}$ and $l(e_1) = a$, $l(e_2) = b$, $l(e_3) = a$, $l(e_4) = b$.

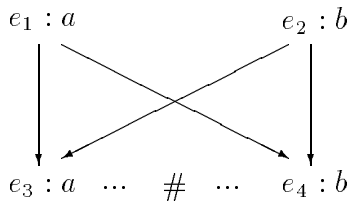


Fig. 1

We will sometimes give algebraic expressions (see [2]) for our examples, to make them easier to understand. The algebraic syntax includes the primitive constructs: sequential composition (;), parallel composition ($\|$), and sum ($+$). The operation ; ($\|$, $+$, respectively) may be easily 'interpreted' by indicating that all events in one component are in the $<$-relation ($\smile$-relation, #-relation, respectively) with all events in the other.

Event structures $\mathcal{E}$ and $\mathcal{F}$ are *isomorphic* ($\mathcal{E} \cong \mathcal{F}$) iff there exists a bijection between their sets of events preserving $<$, # and labelling. An event structure $\mathcal{E}$ is *conflict-free* iff $\#_\varepsilon = \emptyset$.

Isomorphism classes of conflict free event structures are called pomsets (labelled over Act) [17]. Given $a, b \in Act$, we write $a \to b$ for the isomorphism class of $(\{e_1, e_2\}, \{(e_1, e_2)\}, \emptyset, \{(e_1, a), (e_2, b)\})$ and $a|b$ for the isomorphism class of $(\{e_1, e_2\}, \emptyset, \emptyset, \{(e_1, a), (e_2, b)\})$. We use $pom(Act)$ to denote the set of pomsets over Act.

The states of an event structure are called configurations. An event can occur in a configuration only if all the events in its past have occurred. Two events that are in conflict can never both occur in the same stretch of behaviour. Before formalizing the notion of a configuration it will be convenient to adopt the following notation. Let $\mathcal{E}$ be an event structure and $C \subseteq E_\varepsilon$. Then $\downarrow C = \{e \in E_\varepsilon \mid \exists e' \in C \circ e \leq_\varepsilon e'\}$. C is said to be a *configuration* of $\mathcal{E}$ iff $C = \downarrow C$ (left-closed) and $\#_\varepsilon \cap (C \times C) = \emptyset$ (conflict-free). Let $\mathcal{C}(\mathcal{E})$ denote the set of all configurations of $\mathcal{E}$.

Assume $\mathcal{E}$ to be an event structure and $C' \subseteq C \in \mathcal{C}(\mathcal{E})$. Then the *restriction* of $\mathcal{E}$ to C' is defined as $\mathcal{E} \lceil C' = (C', <_\varepsilon \cap (C' \times C'), \#_\varepsilon \cap (C' \times C'), l_\varepsilon |_{C'})$. We denote by C' not only the set itself, but also the labelled partial order it induces by restricting $<_\varepsilon$ and l_ε to C'. It will (hopefully) be clear from the context what is meant. We use $pom_\varepsilon(C) = \{\mathcal{E} \lceil (C'' \setminus C) \mid C \subseteq C'' \in \mathcal{C}(\mathcal{E})\}$ to denote the set of *pomsets* of C.

Definition 2.2. Let $C, C' \in \mathcal{C}(\mathcal{E})$. Then

- $C \rightarrow_\varepsilon C' \stackrel{def}{\Longleftrightarrow} C \subseteq C'$;
- $C \stackrel{p}{\rightarrow}_\varepsilon C' \stackrel{def}{\Longleftrightarrow} C \rightarrow_\varepsilon C'$ and $C' \setminus C = p$ where $p \in pom_\varepsilon(C)$;
- $C \uparrow_\varepsilon C' \stackrel{def}{\Longleftrightarrow} \exists C'' \in C(\mathcal{E}) \diamond (C \rightarrow_\varepsilon C'' \;\&\; C' \rightarrow_\varepsilon C'')$;
- $C \not\uparrow_\varepsilon C' \stackrel{def}{\Longleftrightarrow} \neg(C \uparrow_\varepsilon C')$ (incompatibility);
- $C \uparrow'_\varepsilon C' \stackrel{def}{\Longleftrightarrow} (C \uparrow_\varepsilon C') \;\&\; \neg(C \rightarrow_\varepsilon C' \;\vee\; C' \rightarrow_\varepsilon C)$ (independence). ·

As an illustration, we consider the relations on configurations of the event structure shown in Fig 1.: $\emptyset \stackrel{a|b}{\rightarrow} \{e_1, e_2\}$, $\{e_1\} \uparrow \{e_2\}$, $\{e_1, e_2, e_3\} \not\uparrow \{e_1, e_2, e_4\}$, $\{e_1\} \uparrow' \{e_2\}$.

Lemma 2.1. Let $C, C' \in \mathcal{C}(\mathcal{E})$ Then

(a) $C \uparrow_\varepsilon C' \iff C \cup C' \in \mathcal{C}(\mathcal{E})$;

(b) $C \not\uparrow_\varepsilon C' \iff \exists e \in C \; \exists e' \in C' \diamond e \;\#_\varepsilon\; e'$;

(c) $C \uparrow'_\varepsilon C' \iff \neg(C \subseteq C' \vee C' \subseteq C) \;\&$
$\& \; \forall e \in (C \setminus C') \; \forall e' \in (C' \setminus C) \;\diamond e \smile_\varepsilon e'$. ·

An event structure $\mathcal{E}$ is said to be *without autoconcurrency*, if $\forall e, e' \in E_\varepsilon \diamond ((e \; co_\varepsilon \; e' \;\&\; l_\varepsilon(e) = l_\varepsilon(e')) \Rightarrow e = e')$.

In the following, we will consider only event structures without autoconcurrency and will denote them by the symbols $\mathcal{E}$, $\mathcal{F}$, ...

3 Behavioural Equivalences

In this section, we first introduce some new variants of interleaving and history preserving bisimulations which explicitly express all the relations between events in the structures and therefore nicely fit the model under consideration. These equivalences are then compared, resulting in a lattice of implications.

Definition 3.1. Let $\mathcal{B} \subseteq (\mathcal{C}(\mathcal{E}) \times \mathcal{C}(\mathcal{F})) \cup (\mathcal{C}(\mathcal{F}) \times \mathcal{C}(\mathcal{E}))$ be a symmetric relation, $\alpha \in \{i, h\}$ and $\beta \in \{a, b, c\}^*$. Then

(i) $\mathcal{B}$ is an α-*bisimulation* iff $(\emptyset, \emptyset) \in \mathcal{B}$ and for all $(C, D) \in \mathcal{B}$:
- $\mathcal{E} \lceil C \cong \mathcal{F} \lceil D$, if $\alpha = h$,
- if $C \stackrel{a}{\rightarrow}_\varepsilon C'$ then there is D' such that $D \stackrel{a}{\rightarrow}_\mathcal{F} D'$ and $(C', D') \in \mathcal{B}$.

(ii) $\mathcal{B}$ is an αb-*bisimulation* iff $\mathcal{B}$ is an α-bisimulation and for all $(C, D) \in \mathcal{B}$:
if $C' \stackrel{a}{\rightarrow}_\varepsilon C$ then there is D' such that $D' \stackrel{a}{\rightarrow}_\mathcal{F} D$ and $(C', D') \in \mathcal{B}$.

(iii) $\mathcal{B}$ is an *αa-bisimulation* iff $\mathcal{B}$ is an α-bisimulation and for all $(C, D) \in \mathcal{B}$:
if $C \not\uparrow_{\varepsilon} C'$ then there is D' such that $D \not\uparrow_{\mathcal{F}} D'$ and $(C', D') \in \mathcal{B}$;

(iv) $\mathcal{B}$ is an *αc-bisimulation* iff $\mathcal{B}$ is an α-bisimulation and for all $(C, D) \in \mathcal{B}$:
if $C \uparrow'_{\varepsilon} C'$ then there is D' such that $D \uparrow'_{\mathcal{F}} D'$ and $(C', D') \in \mathcal{B}$.

$\mathcal{E}$ and $\mathcal{F}$ are *$\alpha\beta$-bisimilar*, denoted $\mathcal{E} \approx_{\alpha\beta} \mathcal{F}$, if there exists an $\alpha\beta$-bisimulation $\mathcal{B}$ that is an $\alpha\beta'$-bisimulation for all $\beta' \in \beta$. ·

It is worth noting that instead of defining history preserving variants of bisimulation we introduced mixed-ordering equivalences [5]. This is a possible way because [1] shows that these equivalences coincide in the setting of (labelled) event structures.

We now turn our attention to showing how the bisimulation equivalences defined prior to that are related.

Proposition 3.1. Let $\alpha \in \{i, h\}$, $\beta \in \{a, c\}^*$ and $\beta' \in \{a\}^*$. Then

(a) $\mathcal{E} \approx_{i\beta b} \mathcal{F} \iff \mathcal{E} \approx_{h\beta b} \mathcal{F}$;

(b) $\mathcal{E} \approx_{\alpha\beta' b} \mathcal{F} \iff \mathcal{E} \approx_{h\beta' c} \mathcal{F}$.

(c) $\mathcal{E} \approx_{\alpha\beta' b} \mathcal{F} \iff \mathcal{E} \approx_{\alpha\beta' bc} \mathcal{F}$. ·

Theorem 3.1. Let $\gamma, \delta \in \bigcup_{\beta \in \{a,b,c\}^*} \{\alpha\beta \mid \alpha \in \{i, h\}\}$. Then the following holds: $\mathcal{E} \approx_{\gamma} \mathcal{F}$ implies $\mathcal{E} \approx_{\delta} \mathcal{F}$ iff there is a directed path from $\approx_{\gamma}$ to $\approx_{\delta}$ in Fig. 2. ·

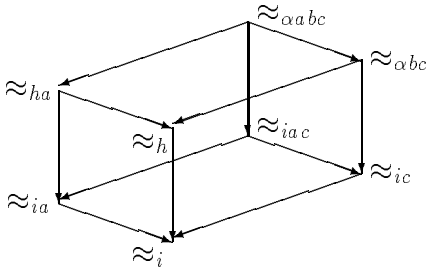


Fig. 2

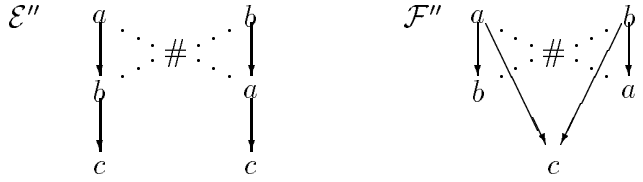
Proof. '$\Leftarrow$' All the implications in Fig. 2 follow from Definition 3.1 and Proposition 3.1.
'$\Rightarrow$' We now show that it is impossible to draw any arrow from one equivalence to the other such that there is no directed path from the first equivalence to the second one in the graph in Fig. 2. For this purpose, we give the following counterexamples.

The event structures $\mathcal{E}_1 = a;(b \,\#\, b)$ and $\mathcal{F}_1 = a;b$ are hb-bisimilar, but they are not ia-bisimilar, because only in $\mathcal{F}_1$ there are no incompatible configurations.

The structures $\mathcal{E}_2 = ((a \parallel (b + c)) + (a \parallel b)) + (b \parallel (a + c))$ and $\mathcal{F}_2 = ((a \parallel (b + c)) + (b \parallel (a + c))) + ((a \parallel (b + c)) + (b \parallel (a + c)))$ are ha-bisimilar. Whereas these structures are not ic-bisimilar, because only in $\mathcal{E}_2$ there exist configurations

consisting of either an event labelled by an a or an event labelled by a b which are not independent from any configuration consisting of an event labelled by a c.

Let us first consider the event structures $\mathcal{E}''$ and $\mathcal{F}''$:



The composed structures $\mathcal{E}_3 = \mathcal{E}'' + \mathcal{F}''$ and $\mathcal{F}_3 = \mathcal{E}'' + \mathcal{E}''$ are iac-bisimilar. Whereas they are not h-bisimilar, because a configuration of $\mathcal{E}_3$, consisting of three events labelled by a, b and c can be related to only a configuration of $\mathcal{F}_3$, also consisting of three events labelled by a, b and c, but these configurations are not isomorphic. ·

4 Logical Characterizations of Equivalences

In this section, we extend the CTL^*-family of logics by introducing some new variants: CTL^*_b with past combinators, CTL^*_a with a conflict modality, CTL^*_c with a concurrency modality, and CTL^*_{abc} that is a combination of the mentioned logics. These logics are further proved to be characteristic for the bisimulations considered above.

We first introduce the syntax and semantics of the most complicated logic CTL^*_{abc}. We define the syntax of CTL^*_{abc} by the following grammar, where we let λ and μ range over CTL^*_{abc} formulas and p range over $pom(Act)$.

$$\lambda, \mu ::= \quad p \mid \quad \neg\lambda \mid \quad \lambda \vee \mu \mid \quad \mu \mathbf{U} \lambda \mid \quad \mathbf{X}\lambda \mid \quad \exists\lambda \quad \mid \quad \mu \mathbf{S} \lambda \mid \quad \mathbf{X}^{-1}\lambda \mid \quad \exists^{-1}\lambda \mid \quad \mathbf{A}\lambda \quad \mid \quad \mathbf{C}\lambda.$$

$$\qquad (1) \quad (2) \qquad (3) \qquad (4) \qquad (5) \quad (6) \qquad (7) \qquad (8) \qquad (9) \qquad (10) \quad (11)$$

Here $\mathbf{S}$ is the "Since" combinator, a past variant of $\mathbf{U}$ ("Until"). $\mathbf{X}^{-1}$ is the immediate past modality, a past variant of $\mathbf{X}$ (immediate future). $\exists^{-1}$ is the "branching past" modality, a past variant of $\exists$ ("branching future"[11]). $\mathbf{C}$ and $\mathbf{A}$ capture concurrency and alternative choice (conflict), respectively.

In the following, we will need some additional notions and notations. A *path* in $\mathcal{E}$ is a sequence of configurations $C_0C_1\ldots$ such that $C_i \stackrel{a_i}{\rightarrow}_\varepsilon C_{i+1}$ with $i = 0, 1, \ldots$. A *run* in $\mathcal{E}$ is its maximal path. We write $\Pi(\mathcal{E})$ for the set of all runs in $\mathcal{E}$. For any $i \geq 0$ and $\pi \in \Pi(\mathcal{E})$, we let $\pi(i) \stackrel{def}{=} C_i$, $\pi^i \stackrel{def}{=} C_iC_{i+1}\ldots$ and $\pi\mid_i \stackrel{def}{=} C_0C_1\ldots C_{i-1}$. Note that $C_0 = \emptyset$ for all $\pi = C_0C_1\ldots \in \Pi(\mathcal{E})$.

As in the original CTL^* logic [6], a CTL^*_{abc} formula expresses properties of some moment in a run of a given $\mathcal{E}$. Formally, the notion of a CTL^*_{abc}-formula λ being satisfied in a run $\pi \in \Pi(\mathcal{E})$ at moment $n = 0, 1, \ldots$ (written $\pi, n \models_{CTL^*_{abc}} \lambda$) is defined by induction on the

length of λ as follows:

$$
\begin{array}{lll}
(1) & \pi, n \models_{CTL^*_{abc}} p & \Longleftrightarrow \exists \pi' \in \Pi(\mathcal{E}), n' \ \diamond \ \pi'(n') \stackrel{p}{\rightarrow} \pi(n). \\
(2) & \pi, n \models_{CTL^*_{abc}} \neg\lambda & \Longleftrightarrow \pi, n \not\models_{CTL^*_{abc}} \lambda; \\
(3) & \pi, n \models_{CTL^*_{abc}} \lambda \vee \mu & \Longleftrightarrow \pi, n \models_{CTL^*_{abc}} \lambda \text{ or } \pi, n \models_{CTL^*_{abc}} \mu; \\
(4) & \pi, n \models_{CTL^*_{abc}} \lambda \mathbf{U} \mu & \Longleftrightarrow \exists k \geq n \ \diamond \ \pi, k \models_{CTL^*_{abc}} \mu \ \& \ \pi, i \models_{CTL^*_{abc}} \lambda \\
 & & \quad \text{for all } n \leq i < k; \\
(5) & \pi, n \models_{CTL^*_{abc}} \mathbf{X}\lambda & \Longleftrightarrow \pi, n+1 \models_{CTL^*_{abc}} \lambda; \\
(6) & \pi, n \models_{CTL^*_{abc}} \exists\lambda & \Longleftrightarrow \exists \pi' \in \Pi(\mathcal{E}) \ \diamond \ \pi' \mid_n = \pi \mid_n \ \& \ \pi', n \models_{CTL^*_{abc}} \lambda; \\
(7) & \pi, n \models_{CTL^*_{abc}} \lambda \mathbf{S} \mu & \Longleftrightarrow \exists 0 \leq k \leq n \ \diamond \ \pi, k \models_{CTL^*_{abc}} \mu \ \& \ \pi, i \models_{CTL^*_{abc}} \lambda \\
 & & \quad \text{for all } k < i \leq n; \\
(8) & \pi, n \models_{CTL^*_{abc}} \mathbf{X^{-1}}\lambda & \Longleftrightarrow n > 0 \ \& \ \pi, n-1 \models_{CTL^*_{abc}} \lambda; \\
(9) & \pi, n \models_{CTL^*_{abc}} \exists^{-1}\lambda & \Longleftrightarrow \exists \pi' \in \Pi(\mathcal{E}) \ \diamond \ \pi'^n = \pi^n \ \& \ \pi', n \models_{CTL^*_{abc}} \lambda; \\
(10) & \pi, n \models_{CTL^*_{abc}} \mathbf{A}\lambda & \Longleftrightarrow \exists \pi' \in \Pi(\mathcal{E}), n' \ \diamond \\
 & & \quad \pi'(n') \not\Diamond_\varepsilon \pi(n) \ \& \ \pi', n' \models_{CTL^*_{abc}} \lambda; \\
(11) & \pi, n \models_{CTL^*_{abc}} \mathbf{C}\lambda & \Longleftrightarrow \exists \pi' \in \Pi(\mathcal{E}), n' \ \diamond \\
 & & \quad \pi'(n') \uparrow'_\varepsilon \pi(n) \ \& \ \pi', n' \models_{CTL^*_{abc}} \lambda.
\end{array}
$$

Informally, $\pi(n)$ is the present, prefix $\pi\mid_n$ is a selected past and π^n is a selected future. A formula p means "atomic property p holds at the present if there exists a past time along some run π' such that from this time up to now p has been performed". $\lambda\mathbf{U}\mu$ means "μ will holds at some point in the future, and λ holds in the meantime", $\lambda\mathbf{S}\mu$ means "μ did holds in the past, and λ has been holding ever since the moment". $\mathbf{X}\lambda$ means "λ holds at the next moment", $\mathbf{X^{-1}}\lambda$ means "λ did hold at the previous moment". $\exists\lambda$ means "the present admits a possible future for which λ holds", $\exists^{-1}\lambda$ means "the present admits a possible past for which λ holds". $\mathbf{A}\lambda$ means "λ holds along some run π' and at some n' such that $\pi'(n')$ is incompatible with $\pi(n)$". $\mathbf{C}\lambda$ means "λ holds along some run π' and at some n' such that $\pi'(n')$ is independent of $\pi(n)$".
It is worth noting that the combinator $\exists^{-1}\lambda$ allows the specification of several interleaving pasts of any time instant, whereas the backward combinators usually considered in the literature (see, for example, [9, 11]) restrict themselves to dealing with a single past of a time instant.

We use the standard abbreviations $\top$, $\bot$, $\wedge$, $\supset$ and $\equiv$, defined in terms of $\neg$ and $\vee$. Moreover, we define: $\mathbf{F}\lambda \stackrel{def}{=} \top\mathbf{U}\lambda$; $\mathbf{G}\lambda \stackrel{def}{=} \neg\mathbf{F}\neg\lambda$; $\mathbf{F^{-1}}\lambda \stackrel{def}{=} \top\mathbf{S}\lambda$; $\mathbf{G^{-1}}\lambda \stackrel{def}{=} \neg\mathbf{F^{-1}}\neg\lambda$; $\forall\lambda \stackrel{def}{=} \neg\exists\neg\lambda$; $\forall^{-1}\lambda \stackrel{def}{=} \neg\exists^{-1}\neg\lambda$; $\bar{\mathbf{C}}\lambda \stackrel{def}{=} \neg\mathbf{C}\neg\lambda$; $\bar{\mathbf{A}}\lambda \stackrel{def}{=} \neg\mathbf{A}\neg\lambda$.

The syntax of CTL_b (CTL_a, CTL_c, respectively) is a restriction of that of CTL^*_{abc}, given by (1) – (9) ((1) – (8) and (10), (1) – (8) and (11), respectively). In the following, we will need relevant fragments of CTL^*_β, denoted by ${}_0CTL^*_\beta$, for $\beta \in \{a, b, c, abc\}$, where only actions from Act are allowed as atomic propositions.

From now on, we use $\Phi({}_\gamma CTL^*_\beta)$ to denote the set of all ${}_\gamma CTL^*_\beta$-formulas, for $\beta \in \{a, b, c, abc\}$ and $\gamma \in \{., 0\}$ (the symbol '.' denotes 'nothing').

We next define some additional satisfiability notions. Let $\lambda \in \Phi(CTL^*_{abc})$ and $C \in \mathcal{C}(\mathcal{E})$. Then λ is called to be

- *satisfiable in* C, — denoted $C \models_{\gamma CTL^*_\beta} \lambda$ — iff $\pi, n \models_{\gamma CTL^*_\beta} \lambda$ for all $\pi \in \Pi(\mathcal{E})$ and n such that $\pi(n) = C$;
- *valid in* $\mathcal{E}$, — denoted $\mathcal{E} \models_{\gamma CTL^*_\beta} \lambda$ — iff $C \models_{\gamma CTL^*_\beta} \lambda$ for all $C \in \mathcal{C}(\mathcal{E})$.

The modal equivalence imposed by the logic ${}_\gamma CTL^*_\beta$ is defined as follows:

$$\mathcal{E} \sim_{\gamma CTL^*_\beta} \mathcal{F} \stackrel{def}{\iff} (\mathcal{E} \models_{\gamma CTL^*_\beta} \lambda \text{ iff } \mathcal{F} \models_{\gamma CTL^*_\beta} \lambda) \text{ for all } \lambda \in \Phi({}_\gamma CTL^*_\beta).$$

We finally establish the main result of the paper. Before doing so, we need to introduce the following notion. $\mathcal{E}$ is called to be *autoconflict finite* iff every set of pairwise conflicting events, labelled by the same action, is finite.

Theorem 4.1. Let $\mathcal{E}$ and $\mathcal{F}$ be autoconflict finite and $\beta \in \{a, b, c, abc\}$. Then

(a) $\mathcal{E} \approx_{i\beta} \mathcal{F} \iff \mathcal{E} \sim_{{}_0 CTL^*_\beta} \mathcal{F}$,

(b) $\mathcal{E} \approx_{h\beta} \mathcal{F} \iff \mathcal{E} \sim_{CTL^*_\beta} \mathcal{F}$. .

We now give some illustrations for the concepts introduced in this section. The non-ia-bisimilar event structures $\mathcal{E}_1$ and $\mathcal{F}_1$ (see above) are distinguished by the formula $a \to b \supset \mathbf{A}\ a \to b$ which only holds of $\mathcal{E}_1$. We next consider the event structures $\mathcal{E}_2$ and $\mathcal{F}_2$ (see above). Since $\mathcal{E}_2 \not\approx_{ic} \mathcal{F}_2$, there must be a formula distinguishing them. Indeed, take $\lambda = a \wedge \neg \mathbf{C}c \supset \mathbf{C}(b \wedge \mathbf{C}c)$. Then $\mathcal{E}_2 \not\models \lambda$ and $\mathcal{F}_2 \models \lambda$. Using the non h-bisimilar event structures $\mathcal{E}_3$ and $\mathcal{F}_3$ (see above) and $\lambda = c \supset a \to b \to c \ \vee \ b \to a \to c$, we then have $\mathcal{E}_3 \models \lambda$ and $\mathcal{F}_3 \not\models \lambda$.

5 Concluding Remarks

We have introduced some new notions of bisimulation which respect all the relations – causality, concurrency, and nondeterminism – between events of distributed systems. We have given concrete characterizations of the bisimulations on event structures. The close interrelations between these equivalences have been established, resulting in a lattice of implications. We have also characterized the proposed bisimulations logically. To this end, we have introduced some new CTL^* like logics with modalities expressing concurrency and conflict, in addition to past and future modalities. These logics provide not only a better understanding of the behavioural equivalences but also natural formal languages to reason about the behaviour of distributed systems. noninterleaving bisimulations.

We hope this article and [19] demonstrate that bisimulations and temporal logics based on the semantics of concurrency and nondeterminism deserve further study. It also deserves to be broadened: as a point in case, we have to mention the decidability of the equivalences proposed and the satisfiability of the logics introduced.

References

[1] Aceto L.: *History Preserving, Causal and Mixed-ordering Equivalence over Stable Event Structures* Fundamenta Informaticae **17(4)** (December 1992) 319 – 331

[2] Boudol, G., Castellani, I.: *Concurrency and Atomicity* Theoretical Computer Science **59** (1988) 25 – 84

[3] Browne, M.C., Clarke, E.M., Grumberg, O.: *Characterizing Finite Kripke Structures in Propositional Temporal Logic* Theoretical Computer Science **59** (1988) 115 – 131

[4] Cherief F.: *Investigations of Back and Forth Bisimulations on Prime Event Structures* Computers and Artificial Intelligence **11** (1992)

[5] Degano, P., deNicola, R., Montanari, U.: *Partial Orderings Descriptions and Observations of Nondeterministic Concurrent Processes* Lecture Notes in Computer Science **354** (1989) 438 – 466

[6] Emerson, E.A., Halper, J.Y.: *"Sometimes" and "Not Never" Revisited: On Branching Versus Linear Time Logic* Journal of the ACM **33(1)** (1986) 151 – 178

[7] Glabbeek, R., Goltz, U.: *Equivalence Notions for Concurrent Systems and Refinement of Action* Lecture Notes in Computer Science **379**(1989) 237 – 248

[8] Goltz, U., Kuiper, R., Penczek, W.: *Propositional Temporal Logics and Equivalences* Lecture Notes in Computer Science **630** (1992) 222 – 236

[9] Hafer, T., Thomas, W.: *Computation Tree Logic CTL* and Path Quantifiers in the Monadic Theory of the Binary Tree* Lecture Notes in Computer Science **267** (1987) 269 – 279

[10] Joyal A., Nielsen, M., Winskel, G.: *Bisimulation and Open Maps* Proc. LICS'93 (1993) 418 – 427

[11] Laroussinie, F., Schnoebelen, Ph.: *A Hierarchy of Temporal Logics with Past* Theoretical Computer Science **148** (1995) 303 – 324

[12] Mukund, M., Thiagarajan, P.S.: *A Logical Axiomatization of Well Branching Event Structures* Theoretical Computer Science **96** (1992) 35 – 72

[13] deNicola, R., Vaandrager, F.: *Three Logics for Branching Bisimulation* Proc. LICS'90 (1990)

[14] Nielsen, M., Plotkin, G., Winskel, G.: *Petri Nets, Event Structures and Domains* Theoretical Computer Science **13** (1981) 85 – 108

[15] PINCHINAT, S.: *Ordinal Processes in Comparative Concurrency Semantics* Lecture Notes in Computer Science **626** (1992)

[16] PINCHINAT, S., LAROUSSINIE, F., SCHNOEBELEN, PH.: *Logical Characterization of Truly Concurrent Bisimulation* Research Report RT 114 IMAG-23 LIFIA (Grenoble, 1994)

[17] PRATT, V.R.: *Modeling Concurrency with Partial Orders* Int. Journal of Parallel Programming **15(1)** (1986) 33 – 71

[18] RABINOVICH, A., TRAKHTENBROT, B.A.: *Behavior Structures and Nets* Fundamenta Informaticae **11** (1988) 357 – 404

[19] VIRBITSKAITE, I., VOTINTSEVA, A.: *Behavioural Characterizations of Partial Order Logics* Proc. FCT'97, Lecture Notes in Computer Science **1279** (1997)

[20] VOGLER, W.: *Bisimulation and Action Refinement* Lecture Notes in Computer Science **480** (1991) 309 – 321

[21] WINSKEL, G.: *An Introduction to Event Structures* Lecture Notes in Computer Science **354** (1989) 364 – 397

Publications in the FI MU Report Series are in general accessible via WWW and anonymous FTP:

`http://www.fi.muni.cz/informatics/reports/`
`ftp ftp.fi.muni.cz (cd pub/reports)`

Copies may be also obtained by contacting:

**Faculty of Informatics
Masaryk University
Botanická 68a
602 00 Brno
Czech Republic**