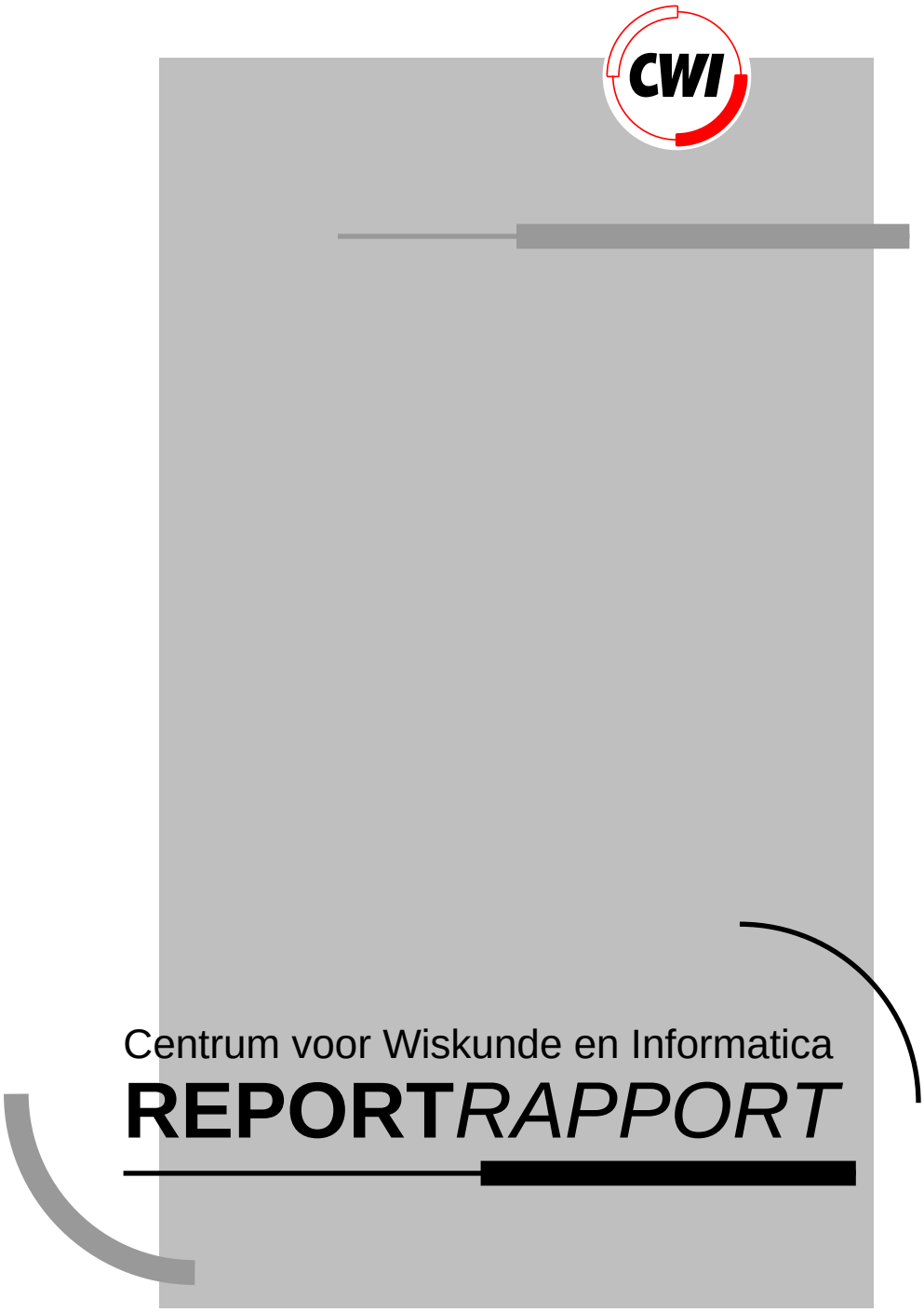

Centrum voor Wiskunde en Informatica

REPORT*RAPPORT*

A theory for simulator tools

H.P. Korver

Computer Science/Department of Software Technology

CS-R9302 1993

A Theory for Simulator Tools

Henri Korver

CWI

P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

henri@cwi.nl

Abstract

This paper presents a possible formalisation of the notion *simulator tool* for process languages like ACP, CCS, (μ)CRL, LOTOS and PSF. First we give precise definitions for the notions *simulator* and *simulation*. Then we can investigate the equivalence that a simulator induces on the explored process terms. This is done by considering two processes, say p and q, equivalent if each simulation of p is also a simulation of q and vice versa. It is proven that there is *no* 'reasonable' simulator inducing bisimulation equivalence. Furthermore it is demonstrated that simulators inducing coarser equivalences, e.g. ready, failure and trace equivalences, are unlikely to be computationally tractable. Our conclusion is that a practical simulator induces an equivalence that is finer (less identifying) than bisimulation and even finer than graph isomorphism.

1991 Mathematics Subject Classification: 68Q55 (Semantics).
1991 CR Categories: D.2.5 (Testing and debugging), F.3.2 (Semantics).
Keywords & Phrases: Head Normalisation, Labelled Transition System, Simulation, Simulator, Structured Operational Semantics.
Note: The author is partly supported by the European Communities under RACE project no. 1046, Specification and Programming Environment for Communication Software (SPECS). This document does not necessarily reflect the view of the SPECS project.

1 Introduction

Nowadays the so-called simulator tools can not be thought away in the validation of concurrent system specifications. For instance a considerable amount of simulator tools have been developed in the area of process algebra. In this setting a simulator can be considered as a tool that is used to explore the underlying state space of process term in a certain language; one can think of simulators for CCS [CPS90], (μ)CRL [SPE92, Ver92], LOTOS [Eer91, vE89, Tre89, vEVD89] or PSF [Vel93]. The basic operation of these simulators is as follows. From a given process term the set of one-step transitions is computed. Subsequently one of these transitions is chosen and the next state, mostly given as a process term again, is returned. Then the whole procedure can be repeated recursively for this next state.

A considerable amount of manpower has already been invested in building simulator tools like the ones mentioned above. But, it is remarkable that (as far as we know) there is no standard theory about simulator tools. For instance, consider the following questions one can ask about simulator tools:

1. What is a simulator (tool), i.e. can we define the notion of a simulator formally? Although there are many simulator tools around now, the question what a simulator actually is has not been answered yet.

2. What is the semantics of a simulator and how does it relate with established process models like the bisimulation model? For example consider the two CCS processes `a.a.0 + a.a.0` and `a.a.0 + a.(a.0 + a.0)` of which it is well-known that they are bisimilar. However a

session with the CWB (Concurrency Workbench) [CPS90] that is displayed in table 1, shows that the simulator does not respect bisimulation semantics. I.e. the CWB does not identify bisimilar processes as already the first menus differ, e.g. `M(a.a.0 + a.a.0)=[1: --- a ---> a.0]` and `M(a.a.0 + a.(a.0 + a.0))= [1: --- a ---> a.0; 2: --- a ---> a.0 + a.0]`. Clearly this simulator is more concrete than bisimulation semantics as it distinguishes two

```
                The Edinburgh Concurrency Workbench
                   (Version 6.1, October 1, 1992)


Command: bi P
Agent: a.a.0 + a.a.0

Command: bi Q
Agent: a.a.0 + a.(a.0 + a.0)

Sim> sim
Agent: P

Simulated agent: P
Transitions:
    1: --- a ---> a.0

Sim> 1
  --- a --->

Simulated agent: a.0
Transitions:
    1: --- a ---> 0

Sim> sim
Agent: Q

Simulated agent: Q
Transitions:
    1: --- a ---> a.0
    2: --- a ---> a.0 + a.0
```

Table 1: A session with the Concurrency Workbench.

a-transitions in the second menu. The question whether there exist practical simulators that make the same identifications as the well-known bisimulation model does, is the main topic of this paper. This question is useful because if the answer is positive then we can implement simulators that do not distinguish more that bisimulation semantics. This would then imply the existence of a simulator that does not confuse us with more details about processes than bisimulation semantics allows.

In this paper, we try to answer the two questions stated above. As a possible answer the first question, we formalise the notion of a simulator by defining it as a triple $Sim = (C, M, R)$ of a *conversion-, menu-* and a *residue* function. The conversion function C maps a process term that is fed into the

simulator to the initial state of its exploration. The menu function computes the set of all possible one-step transitions of a state in the exploration. The residue function R computes the next state when one of the transitions in the menu is chosen. Then a simulation of a term p can be formalised as an alternating sequence of menus offered by the simulator and choices $c_1, c_2, \ldots$ from these menus, i.e. $M(C(p))\ c_1\ M(R(C(p), c_1))\ c_2\ M(R(R(C(p), c_1), c_2))\ \ldots$

As a possible answer to the second question, we define the semantics of a simulator as the equivalence that a simulator $Sim = (C, M, R)$ induces on process terms as follows. The equivalence induced by a simulator is obtained by considering two process terms p and q equivalent exactly when each simulation of p is also a simulation of q and vice versa. We shall prove that in general there do not exist simulators respecting bisimulation equivalence. Furthermore, we conjecture that simulators inducing coarser (trace based) equivalences have a non-polynomial computational complexity. Our conclusion is that the equivalence induced by a practical simulator must be finer than bisimulation and even finer than graph isomorphism.

2 Acknowledgements

I am very grateful to Jan Bergstra for being the person really understanding and backing up my ideas. Without his help and mental support this paper would have never reached its current form.

3 Preliminaries

3.1 Labelled Transition Systems

In this paper, we restrict ourselves to the 'interleaving paradigm' and consider labelled transition systems as the basic model for processes.

Definition 3.1. A *labelled transition system* (LTS) is a 4-tuple $T = (S, L, \longrightarrow, r)$, where:

- S is a set of states;
- L is a set of transition labels;
- $\rightarrow \subseteq S \times L \times S \cup \{\top\}$ is the transition relations where $\top$ is a distinguished element called the *termination* state;
- r is the *root* state.

The domain of LTS's is denoted by LTS. □

An element $(s, a, s') \in \longrightarrow$ is called a transition, and is usually written in a more 'pictorial' notation $s \xrightarrow{a} s'$. In this pictorial notation, the arrow $\longrightarrow$ is also called an *edge*. We shall use the following notations for transitions.

Notation 3.2. (1) $s \xrightarrow{a} s'$ for $s \xrightarrow{a} s' \in \longrightarrow$ (2) $s \xrightarrow{a}$ for $\exists s' : s \xrightarrow{a} s'$ (3) $s \not\xrightarrow{a}$ for not $s \xrightarrow{a}$ (4) $s \not\rightarrow$ for $\forall a \in L : s \not\xrightarrow{a}$ (5) $s \xrightarrow{a_1 \cdots a_n} s'$ for $s \xrightarrow{a_1} \xrightarrow{a_2} \cdots \xrightarrow{a_n} s'$. □

A state s is called a *termination* state if $s \equiv \top$. A state s is called a *deadlock* state if $s \not\equiv \top$ and $s \not\rightarrow$.

Definition 3.3. *(Isomorphism.)* Two LTS's: $g, h \in LTS$ are *isomorphic*, notation $g \simeq h$, if there exists a bijective mapping between their sets of states which preserves *roots, termination states* ($\top$) and *transitions.* □

The notion of bisimulation equivalence plays a central role in this paper.

Definition 3.4. *(Bisimulation.)* Let $g_i = (S_i, L_i, \longrightarrow_i, r_i)\,(i = 1, 2)$ be LTS's. A relation $R \subseteq S_1 \times S_2$ is a *(strong) bisimulation* between g_1 and g_2 if it satisfies:

- $r_1 R\, r_2$;
- – if sRt and $s \xrightarrow{a}_1 s'$, then there is a $t' \in S_2$ with $t \xrightarrow{a}_2 t'$ and $s'Rt'$.
  – if sRt and $t \xrightarrow{a}_2 t'$, then there is a $s' \in S_2$ with $s \xrightarrow{a}_1 s'$ and $s'Rt'$.
- – if sRt and $s \xrightarrow{a}_1 \top$, then $t \xrightarrow{a}_2 \top$.
  – if sRt and $t \xrightarrow{a}_2 \top$, then $s \xrightarrow{a}_1 \top$.

LTS's g_1 and g_2 are *bisimilar,* notation $g_1 \leftrightarrows g_2$, if there is a bisimulation between them. Note that bisimilarity is an equivalence relation. □

3.2 A Process Specification Language

As a running example, we here present the language of ACP [BW90]. ACP is chosen as one of the many algebraic formalisms for specifying processes (LTS's), like CCS, CSP and MEIJE, because I am the most familiar with it. But of course the other candidates can be used equally well.

We consider the following ACP syntax:

$p := \delta \mid p \cdot p \mid p + p \mid p \parallel p \mid a \mid \delta_H(p) \mid x$ where

- a ranges over a *finite* set of actions A.
- $H \subseteq A$.
- x ranges over a *finite* set of constants $Cons = \{X, Y, \ldots\}$. A constant is defined by a, possibly recursive, *constant definition* $x \stackrel{\text{def}}{=} p$. A set Δ of constant definitions is called a *constant declaration.* We here assume that, given a constant definition $x \stackrel{\text{def}}{=} p$, each occurrence of x in p is *guarded,* i.e. x occurs only within subterms of the form $a \cdot q$ of p. The set $Cons(\Delta)$ contains the constants appearing in a declaration Δ.

The set of ACP terms generated by p is denoted by $Terms$(ACP) or just $Terms$ when clear from the context. A generic process is denoted by $p, q, \ldots$ The operators to built process terms are sequential composition $p \cdot p$, summation $p + p$, parallel composition $p \parallel p$ and encapsulation $\delta_H(p)$ which renames actions in H into δ. The constant δ stands for deadlock. In a product $p \cdot q$ we will often omit the 'dot' $(\cdot)$. We take sequential composition $(\cdot)$ to be more binding than other operations and $+$ to be less binding than other operations. In case we are dealing with an associative operator, we also leave out the parentheses.

Definition 3.5. *(The LTS specified by an ACP term.)* Suppose that an action set A, a communication function $\gamma : A \times A \to A$ and a (recursive) constant declaration Δ are given. Then let $\longrightarrow$ be the transition relation defined by the action rules given in table 2. A term $p \in Terms$(ACP) *specifies* the LTS: $SOS(p) \stackrel{\text{def}}{=} (Terms(\text{ACP}), A, \longrightarrow, p)$. □

Usually, action rules like the ones presented in table 2 follow the structure of the process terms and are therefore called SOS (acronym for Structured Operational Semantics) rules [Plo81].

Definition 3.6. *(An LTS equivalence class model for ACP.)* Let $\sim$ be a given equivalence relation on LTS. The equivalence class $\{g \in LTS \mid g \sim SOS(p)\}$ is the interpretation of an ACP term $p \in Terms$(ACP) in the model $LTS/\sim$. □

$a \in A$	:	$a \xrightarrow{a} \top$
$\cdot$	:	$\dfrac{p \xrightarrow{a} \top}{p \cdot q \xrightarrow{a} q}$ $\qquad$ $\dfrac{p \xrightarrow{a} p'}{p \cdot q \xrightarrow{a} p' \cdot q}$
$+$	:	$\dfrac{p \xrightarrow{a} \top}{p + q \xrightarrow{a} \top \quad q + p \xrightarrow{a} \top}$ $\qquad$ $\dfrac{p \xrightarrow{a} p'}{p + q \xrightarrow{a} p' \quad q + p \xrightarrow{a} p'}$
$\parallel$	:	$\dfrac{p \xrightarrow{a} p'}{p \parallel q \xrightarrow{a} p' \parallel q \quad q \parallel p \xrightarrow{a} q \parallel p'}$
		$\dfrac{p \xrightarrow{a} \top}{p \parallel q \xrightarrow{a} q \quad q \parallel p \xrightarrow{a} q}$
If $\gamma(a,b) = c$, then		
$\parallel$	:	$\dfrac{p \xrightarrow{a} p' \quad q \xrightarrow{b} q'}{p \parallel q \xrightarrow{c} p' \parallel q'}$ $\qquad$ $\dfrac{p \xrightarrow{a} \top \quad q \xrightarrow{b} \top}{p \parallel q \xrightarrow{c} \top}$
		$\dfrac{p \xrightarrow{a} \top \quad q \xrightarrow{b} q'}{p \parallel q \xrightarrow{c} q' \quad q \parallel p \xrightarrow{c} q'}$
∂_H	:	$\dfrac{p \xrightarrow{a} \top \quad a \notin H}{\partial_H(p) \xrightarrow{a} \top}$ $\qquad$ $\dfrac{p \xrightarrow{a} p' \quad a \notin H}{\partial_H(p) \xrightarrow{a} \partial_H(p')}$
Recursion	:	$\dfrac{p \xrightarrow{a} \top \quad (X \stackrel{\text{def}}{=} p) \in \Delta}{X \xrightarrow{a} \top}$ $\qquad$ $\dfrac{p \xrightarrow{a} p' \quad (X \stackrel{\text{def}}{=} p) \in \Delta}{X \xrightarrow{a} p'}$

Table 2: SOS rules for *Terms*(ACP).

4 A Definition for a Simulator

In definition 4.1 below we formalise a simulator as a 3-tuple $Sim = (C, B, R)$ consisting of a conversion-, branching- (menu) and a residue function. For technical convenience, we shall define the menu function via the more compact branching function.

Definition 4.1. *(Simulator.)* Let $\mathsf{L} = \{p^1, p^2, \ldots\}$ be an enumerable set of process terms, let $\mathsf{Act} = \{a^1, a^2, \ldots\}$ be an enumerable set of actions and let $\mathsf{S} = \{s^1, s^2, \ldots\}$ be an enumerable set of states. The function triple $Sim = (C, B, R)$ where

$$\begin{array}{ll} C : \mathsf{L} \to \mathsf{S} & \text{(Conversion function)} \\ B : \mathsf{S} \times \mathsf{Act} \to \mathbb{N} & \text{(Branching function)} \\ R : \mathsf{S} \times \mathsf{Act} \times \mathbb{N} \to \mathsf{S} \cup \{\bot, \top\} & \text{(Residue function)} \end{array}$$

is a *simulator* (for language L) if

- C, B and R are *computable* functions, i.e. *(total) recursive* (in the sense of [Rog67]) with respect to the enumerations of L, Act and S.
- For each $s \in S$ the *menu* set $M(s) := \{a_i \in \mathsf{Act} \times \mathbb{N} \mid 0 < i \leq B(s, a)\}$ can be computed *effectively*. That is, there is a function $M : \mathbb{N} \to \mathbb{N}$ defined by

  $$M(k) \stackrel{\text{def}}{=} CI(\{\langle l, i\rangle \mid 0 < i \leq B(s^k, a^l)\})$$

  that is *recursive* with respect to the enumerations of Act and S.[1]
- $a_i \notin M(s) \iff R(s, a, i) = \bot$ for all $s \in \mathsf{S}$, $a \in \mathsf{Act}$ and $i \in \mathbb{N}$.

□

The intuition of the functions C, B, R is as follows. The *conversion* function $C : \mathsf{L} \to \mathsf{S}$ maps a process term in L to its initial (root) state in S.

The *branching* function $B : \mathsf{S} \times \mathsf{Act} \to \mathbb{N}$ computes, given a state s and an action a, the number of outgoing a-edges of s. The branching function determines a *menu* set $M(s) := \{a_i \in \mathsf{Act} \times \mathbb{N} \mid 0 < i \leq B(s, a)\}$ for each state in S. The menu set $M(s)$ displays all the actions $a \in \mathsf{Act}$ that appear as labels on the outgoing edges of state $s \in S$. These actions are indexed with a natural number $i > 0$ for linking them uniquely to their corresponding edges. This is needed to distinguish actions that are the same but appear on different edges (non-determinism). We assume that the menu set is finite and that it can always be computed in an effective way.

The *residue* function $R : \mathsf{S} \times \mathsf{Act} \times \mathbb{N} \to \mathsf{S} \cup \{\bot, \top\}$ returns, given a state s and an action a, the (*next*) state to which the i^{th} outgoing a-edge of s leads. The range of R includes two special symbols $\bot$ and $\top$. $R(s, a, i) = \bot$ means that the i^{th} outgoing edge of state s is *undefined* (denoted by $\bot$). $R(s, a, i) = \top$ means that the i^{th} outgoing edge of state s leads to a *termination state* denoted by $\top$ (see definition 3.1). If a termination state is reached, the simulator stops the exploration. It is assumed that the residue function is only defined on choices from the menu as is formalised in the definition.

Definition 4.2. *(Polynomial simulator.)* A simulator $Sim = (C, B, R)$ is *polynomial* if the functions C, B, R and M can be computed in time that is polynomial with respect to the size of terms in L, Act, S and $\mathbb{N}$.

□

[1] The canonical index *(CI)* of $\emptyset$ is 0 and of $\{k_1, k_2, \ldots, k_l\}$ it is the number $2^{k_1} + 2^{k_2} + \ldots + s^{k_l}$. An ordered pair $\langle k, l\rangle$ is coded by $\frac{1}{2}(k^2 + 2kl + l^2 + 3k + l)$. See [Rog67].

5 An Example of a Simulator

Most simulator tools [CPS90, Eer91, vE89, Tre89, Vel93, SPE92] for process languages are implemented via action (SOS) rules. An exception is the simulator for μCRL, developed by EMILE VERSCHUREN [Ver92], which is based on rewriting μCRL terms into head normal form.

As an example, we instantiate a simulator for ACP via the SOS rules given in table 2. This example is a simplification but gives the underlying idea of the working of most existing simulator tools.

Simulator 5.1. *(SOS simulator for ACP.)* Let $A = \{a, b\}$ and $\Delta = \{P \stackrel{\text{def}}{=} abX + aa + b\delta,\ X \stackrel{\text{def}}{=} a\}$. Define a simulator $Sim = (C, B, R)$ for $Terms(\text{ACP})$ as follows:

- $\mathsf{S} = \mathsf{L} = Terms(\text{ACP})$, $\mathsf{Act} = A$.
- $C(p) = p$ for all $p \in \mathsf{L}$
- Let $Succ(p, a) := \{p' \mid p \xrightarrow{a} p'\}$ be the a-successor set of state $p \in \mathsf{S}$, where $\longrightarrow$ is defined by the action rules in table 2.
  - $B(p, a) = |Succ(p, a)|$.
  - Let $\prec$: $Terms(\text{ACP}) \times Terms(\text{ACP})$ be an arbitrary but fixed *strict ordering*[2] on ACP terms. Then define the residue function by

$$R(p, a, i) = p^i \quad \text{if } \forall j.\ (i < j \Longleftrightarrow p^i \prec p^j \text{ and } p^i, p^j \in Succ(p, a)).$$

    The intuition is that $R(p, a, i)$ is the i^{th} element in the set $Succ(p, a)$ with respect to the $\prec$-ordering. For easy understanding of the working of this simulator that will be explained below, we assume that the $\prec$-ordering satisfies: $a \prec b \prec X \prec bX$.

□

Now let us try to understand the working of simulator 5.1. Let $P \stackrel{\text{def}}{=} abX + aa + b\delta$ (with $X \stackrel{\text{def}}{=} a$) be a term in L that we want to explore with simulator 5.1. The initial state of process P is given by the term $C(P)$. The menu of $C(P)$ is given by $M(C(P)) = M(P) = \{a_1, a_2, b_1\}$ expressing that the initial state of P has three outgoing edges denoted by a_1, a_2 and b_1. By selecting the edge a_1, the next state is given by $R(P, a, 1) = a$ because $a \prec bX$. By selecting the edge a_2, we have $R(P, a, 2) = bX$. By selecting b_1, we enter a deadlock state: $R(P, b, 1) = \delta$. This procedure can be repeated recursively as pictured in figure 1 until a termination ($\top$) or a deadlock (δ) state is reached.

Without proof we remark that all the functions of simulator 5.1 can be computed in polynomial time.

Proposition 5.2. Simulator 5.1 is *polynomial*. □

6 Defining a Session with a Simulator

A session of a term p with a simulator $Sim = (C, B, R)$ is obtained by selecting an element a_i from the menu $M(C(p))$ and repeating this procedure recursively for the 'next' state given by the state $R(C(p), a, i)$.

Definition 6.1. *(A session with a simulator.)* Let $Sim = (C, B, R)$ be a simulator for L with action set Act and state set S. A *session* φ of a term $p \in \mathsf{L}$ *with* simulator Sim is one of the following alternating sequences:

[2] A strict ordering is a binary relation that is transitive, irreflexive and total.

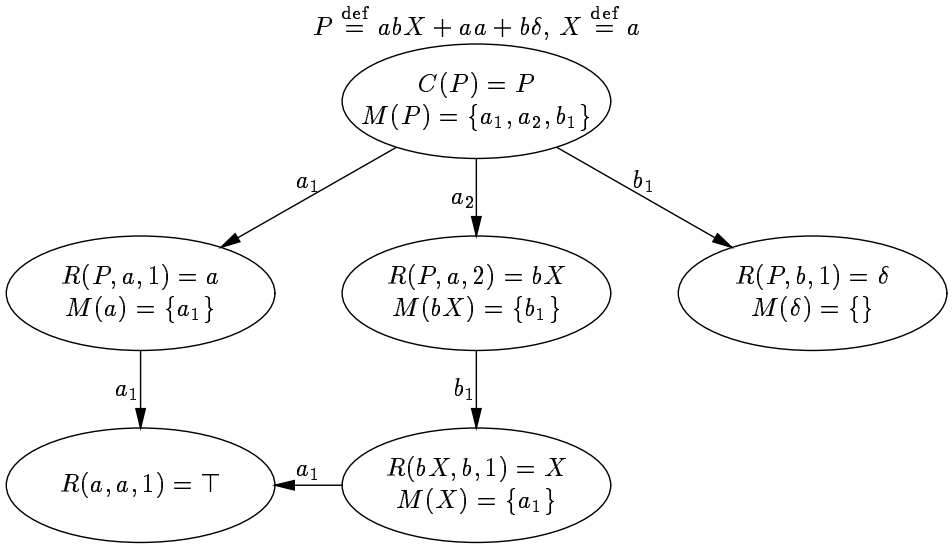

Figure 1: The working of the example simulator.

1. $M(s^0)\ a^0_{i_0}\ M(s^1)\ \ldots\ldots\ M(s^n)$ if $M(s^n) \neq \emptyset$ (*partial session*)
2. $M(s^0)\ a^0_{i_0}\ M(s^1)\ \ldots\ldots\ M(s^n)$ if $M(s^n) = \emptyset$ (*deadlock session*)
3. $M(s^0)\ a^0_{i_0}\ M(s^1)\ \ldots\ldots\ M(s^n)\ a^n_{i_n}$ if $R(s^n, a^n, i^n) = \top$ (*termination session*)

where

- $s^j \in \mathsf{S}$, $a^j_{i_j} \equiv (a^j, i^j) \in (\mathsf{Act} \times \mathbb{N})$, $0 \leq j \leq n$
- $s^0 = C(p)$
- $a^j_{i_j} \in M(s^j)$
- $s^{j+1} = R(s^j, a^j, i^j)$

□

Example 6.2. Below all the simulation sessions of $P \stackrel{\text{def}}{=} abX + aa + b\delta$ (where $X \stackrel{\text{def}}{=} a$) with simulator 5.1 are summerised.

1.	$\{a_1, a_2, b_1\}$	partial
2.	$\{a_1, a_2, b_1\}\ a_1\ \{a_1\}$	partial
3.	$\{a_1, a_2, b_1\}\ a_1\ \{a_1\}\ a_1$	terminated
4.	$\{a_1, a_2, b_1\}\ a_2\ \{b_1\}$	partial
5.	$\{a_1, a_2, b_1\}\ a_2\ \{b_1\}\ b_1\ \{a_1\}$	partial
6.	$\{a_1, a_2, b_1\}\ a_2\ \{b_1\}\ b_1\ \{a_1\}\ a_1$	terminated
7.	$\{a_1, a_2, b_1\}\ b_1\ \{\}$	deadlock

For example, simulation session 7 above expresses that the initial state of the process graph of the process P has two outgoing a-edges and one outgoing b-edge. By selecting the b-edge the process evolves in a deadlock state. *(End example.)*

7 The Semantics of a Simulator

7.1 Sessions and Simulations

Definition 7.1. *(Session set of a term induced by a simulator.)* Let $Sim = (C, B, R)$ be a simulator for L. The *session set* of a process term $p \in \mathsf{L}$ *induced* by Sim is given by

$$SES_{Sim}(p) = \{\varphi \mid \varphi \text{ is a } \textit{simulation session} \text{ of } p \text{ with } Sim\ \}.$$

□

Definition 7.2. *(Equivalence on terms induced by a simulator.)* A simulator $Sim = (C, B, R)$ *induces* an *equivalence* $\equiv_{Sim}$ on L as follows:

$$p \equiv_{Sim} q \iff SES_{Sim}(p) = SES_{Sim}(q) \quad \text{for all } p, q \in \mathsf{L}.$$

□

In the following, we show that if we leave out the menus in a session set we still have the same identifications on process terms.

Definition 7.3. *(Simulation.)* Let φ be a *session* with a simulator. Define a *simulation* $\overline{\varphi}$ as follows:

- $\overline{\varphi} \stackrel{\text{def}}{=} \lambda$ if $\varphi = M(s^0)$.
- $\overline{\varphi} \stackrel{\text{def}}{=} a_{i_0}^0 \ \ldots\ldots\ a_{i_{n-1}}^{n-1}$ if $\varphi = M(s^0)\ a_{i_0}^0\ M(s^1)\ \ldots\ldots\ a_{i_{n-1}}^{n-1}\ M(s^n)$
- $\overline{\varphi} \stackrel{\text{def}}{=} a_{i_0}^0 \ \ldots\ldots\ a_{i_n}^n \dagger$ if $\varphi = M(s^0)\ a_{i_0}^0\ M(s^1)\ \ldots\ldots\ a_{i_{n-1}}^{n-1}\ M(s^n)\ a_{i_n}^n$.

□

In words, a simulation $\overline{\varphi}$ is defined as session φ where the menus are left out. In case φ is a *termination* session, a termination symbol $\dagger$ is appended. The *empty simulation* λ corresponds to the simulation session $\varphi = M(s^0)$ where the menu of the initial state is displayed but no choice has yet been made.

Definition 7.4. *(Simulation set induced by a simulator.)* Let $Sim = (C, B, R)$ be a simulator for L. The *simulation set* of a term $p \in \mathsf{L}$ *induced* by Sim is given by the set

$$SIM_{Sim}(p) = \{\overline{\varphi} \mid \varphi \in SES_{Sim}(p)\}.$$

□

In respect with example 6.2, we have

$$SIM_{Sim}(P) = \{\lambda, a_1, a_1a_1\dagger, a_2, a_2b_1, a_2b_1a_1\dagger, b_1\}.$$

Theorem 7.5. Let $Sim = (C, B, R)$ be a simulator for L. For every $p, q \in \mathsf{L}$, we have that

$$SIM_{Sim}(p) = SIM_{Sim}(q) \iff SES_{Sim}(p) = SES_{Sim}(q).$$

□

7.2 The LTS Explored by the Simulator

Below we give a possible definition of the LTS of a term that is explored by a simulator.

Definition 7.6. *(The LTS of a term explored by a simulator.)* The LTS of a term $p \in \mathsf{L}$ *explored* by a simulator $Sim = (C, B, R)$ is given by $LTS_{Sim}(p) = (\mathsf{S}, \mathsf{Act}, \longrightarrow, (\lambda, C(p)))$ where $\longrightarrow$ is defined by the following rules (one for each $a \in \mathsf{Act}, i \in \mathbb{N}, s \in \mathsf{S}, \sigma \in (\mathsf{Act} \times \mathbb{N})^*$):

$$\begin{array}{ll} (\sigma, s) \xrightarrow{a} \top & \text{if } R(s,a,i) = \top \\ (\sigma, s) \xrightarrow{a} (\sigma a_i, R(s,a,i)) & \text{if } a_i \in M(s) \text{ and } R(s,a,i) \neq \top \end{array}$$

□

This definition shall be used to formalise a semantic criterion for simulators in section 9. Note that $LTS_{Sim}(p)$ is always a tree and therefore we often call it the simulation tree of p.

In figure 2, one can see that a state, say q, in the simulation tree consists of two components $q \equiv (\sigma, s)$. The first component $\sigma \in (\mathsf{Act} \times \mathbb{N})^*$ is the sequence of choices (simulation) leading to q. σ can be considered as a unique identifier for state q. The other component $s \in \mathsf{S}$ is the information for computing the next transitions of q. Note that once the simulation tree is built the second component (s) can be left out.

The LTS of term P explored by simulator 5.1 is given in figure 2.

Corollary 7.7. Let $Sim = (C, B, R)$ be a simulator for L. For every $p, q \in \mathsf{L}$, the following statements are equivalent:

1. $SES_{Sim}(p) = SES_{Sim}(q)$.
2. $SIM_{Sim}(p) = SIM_{Sim}(q)$.
3. $LTS_{Sim}(p) = LTS_{Sim}(q)$.
4. $p \equiv_{Sim} q$ (by definition).

□

8 Soundness and Completeness of a Simulator

In this section we develop the machinery for relating the term-identification of a simulator with the term-identification of the well-known process models which have been developed in process algebra [Gla90]. The following definition says that a simulator Sim is *sound* with respect to a process model $\mathcal{M}$ if the processes identified by Sim are also identified by $\mathcal{M}$.

Definition 8.1. *(Soundness.)* A simulator Sim for a language L is *sound* with respect to a model $\mathcal{M}$ if for all $p, q \in \mathsf{L}$ it holds that $p \equiv_{Sim} q \Rightarrow \mathcal{M} \models p = q$ □

Conversely, we say that a simulator is *complete* with respect to a model $\mathcal{M}$ if the processes identified by $\mathcal{M}$ are also identified by Sim.

Definition 8.2. *(Completeness.)* A simulator Sim for language L is *complete* with respect to a model $\mathcal{M}$ if for all $p, q \in \mathsf{L}$ it holds that $\mathcal{M} \models p = q \Rightarrow p \equiv_{Sim} q$. □

The following definition gives insight in how the SOS simulator from section 4 is related with the graph isomorphism model ($LTS/\simeq$) and the bisimulation model ($LTS/\underline{\leftrightarrow}$).

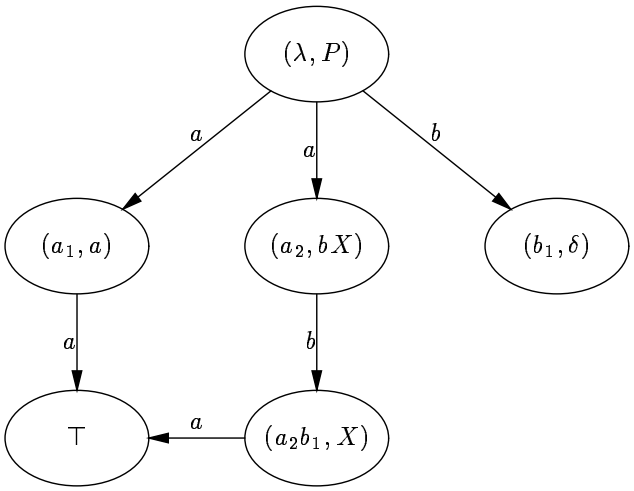

Figure 2: $LTS_{Sim}(P)$ where $P \stackrel{\text{def}}{=} abX + aa + b\delta$, $X \stackrel{\text{def}}{=} a$.

Proposition 8.3. Let *Sim* be simulator 5.1.

1. *Sim* is *sound* with respect to $LTS/\simeq$ as given in definition 3.6.
2. *Sim* is **not** *complete* with respect to $LTS/\simeq$.
3. *Sim* is *sound* with respect to $LTS/\underline{\leftrightarrow}$.
4. *Sim* is **not** *complete* with respect to $LTS/\underline{\leftrightarrow}$.

□

Proof.

1. Straightforward by careful inspection of the definitions of simulator 5.1 and the graph isomorphism model $LTS/\simeq$.
2. $LTS/\simeq \models aX + ab = aa + ab$. But *Sim* distinguishes them: $a_1b_1 \in SIM_{Sim}(aX + ab)$ and $a_1b_1 \notin SIM_{Sim}(aa + ab)$. Note that this distinction is caused by the fact that simulator 5.1 respects the ordering $a \prec b \prec X$.
3. Immediate by 8.3.1 by using the fact that isomorphism is strictly finer than bisimulation (in symbols: $\simeq \subseteq \underline{\leftrightarrow}$).
4. Immediate by 8.3.2. Or as follows: $LTS/\underline{\leftrightarrow} \models aa + aa = aa + a(a + a)$. But *Sim* distinguishes them: $M(aa + aa) = \{a_1\}$ and $M(aa + a(a + a)) = \{a_1, a_2\}$.

□

Proposition 8.3 says that simulator 5.1 induces an equivalence that is strictly finer than graph isomorphism and bisimulation. The following theorem states that in theory there exists a simulator that is sound and complete with respect to bisimulation semantics.

Theorem 8.4. There exists a simulator $Sim = (C, B, R)$ for $Terms(\text{ACP})$ that is *sound* and *complete* with respect to $LTS/\underline{\leftrightarrow}$ as given in definition 3.6. □

We shall proof this theorem by constructing a simulator with the required properties. In doing this we need a projection operator which is given by the following syntax:

$$p := \ldots \mid \pi_n(p) \qquad \text{with } n \in \mathbb{N}.$$

The terms of ACP extended with a projection operator is denoted by $Terms(\text{ACP}, \text{PR})$. The operational semantics of the projection operator is given by the following SOS rules (one for each $n \in \mathbb{N}$ and $a \in A$):

$$\frac{p \xrightarrow{a} p'}{\pi_{n+1}(p) \xrightarrow{a} \pi_n(p')}$$

Intuitively, $\pi_n(p)$ allows p to perform n moves freely, and then stops it. The following lemma is the core of the proof of theorem 8.4.

Lemma 8.5. There exists a *computable* coding function

$$\ulcorner . \urcorner : Terms(\text{ACP}, \text{PR}) \to \mathbb{N}$$

such that for all terms $p, q \in Terms(\text{ACP}, \text{PR})$ and all $n, m \in \mathbb{N}$ it holds that

$$\ulcorner \pi_n(p) \urcorner = \ulcorner \pi_m(q) \urcorner \iff LTS/\underline{\leftrightarrows} \models \pi_n(p) = \pi_m(q).$$

Now we can construct a simulator that is sound and complete with the bisimulation model.

Proof of theorem 8.4. Define a simulator $Sim = (C, B, R)$ for $Terms(\text{ACP})$ as follows:

- Let $\mathsf{S} = Terms(\text{ACP}) \times \mathbb{N}$ and let $\mathsf{Act} = \{a\}$ where a is an arbitrary but fixed action (not necessary in A). A state $(p, n) \in \mathsf{S}$ determines the current projection $\pi_n(p)$ of the simulated process p.
- $C(p) = (p, 1)$. The initial state of p is determined by its first projection $\pi_1(p)$.
- $B((p, n), a) = \ulcorner \pi_n(p) \urcorner$. This definition implies that $M((p, n)) = \{a_1, a_2, \ldots, a_{\ulcorner \pi_n(p) \urcorner}\}$. So, the number of elements in the menu is exactly the encoding of the n^{th} projection of p. By lemma 8.5 we know that this coding takes care that bisimilar terms are identified.
- $R((p, n), a, i) = \begin{pmatrix} (p, n+1) & \text{if } i \leq \ulcorner \pi_n(p) \urcorner \text{ and } \ulcorner \pi_n(p) \urcorner \neq \ulcorner \pi_{n+1}(p) \urcorner \\ \top & \text{if } \ulcorner \pi_n(p) \urcorner = \ulcorner \pi_{n+1}(p) \urcorner \\ \bot & \text{otherwise} \end{pmatrix}$
  The residue function takes care that the depth of the projection is incremented by one in each step of the simulation. Note that a termination symbol ($\top$) is returned when the projection reaches a fixed-point: $\pi_n(p) = \pi_{n+1}(p)$.

□

So, the intuition of the bisimulation simulator is that the cardinality of the menu that is displayed after n steps in the simulation of term p, exactly corresponds to the encoding of lemma 8.5. This encoding takes care that bisimilar terms are identified. It is obvious that the working this simulator goes far beyond any reasonable intuition and in the next session we shall develop an criterion to rule out such simulator definitions.

On a scratch-pad we have proven a similar result as theorem 8.4 for graph isomorphism. However we have not included the proof in this paper because it is very technical and does not really deepen our insights here.

We conjecture that the computational complexity of bisimulation simulators and isomorphism simulators is quite involved as stated below.

Conjecture 8.6. If there exists a *polynomial* simulator Sim for $Terms(\text{ACP})$ that is *sound* and *complete* with respect to $LTS/\underline{\leftrightarrows}$ or $LTS/\simeq$ then NP=P.

9 The Non-Compatibility of a Bisimulation Simulator

We find that the working of the bisimulation simulator given in the previous section goes beyond any 'reasonable' intuition. And therefore we want to rule out such simulator definitions. This can be done by imposing an extra semantic criterion, besides soundness and completeness, on the definition of a simulator as follows.

Definition 9.1. *(Compatibility of a Simulator.)* Let $LTS/\!\sim$ be a model for language L given by the interpretation function $I : \mathsf{L} \to LTS/\!\sim$. A simulator *Sim* for language L is *compatible* with $LTS/\!\sim$ if for all $p \in \mathsf{L}$ it holds that $LTS_{Sim}(p) \in I(p)$.

We say that a simulator *Sim respects* a model $\mathcal{M}$ if *Sim* is *sound, complete* and *compatible* with $\mathcal{M}$. □

This definition says that a simulator is compatible with an equivalence class model $LTS/\!\sim$ if for each simulated term p the corresponding simulation tree $LTS_{Sim}(p)$ is contained in the equivalence class interpretation $I(p)$ of p.

Notation 9.2. We write a^n for the n-fold sequential composition of an $a \in A$ with itself: $a \cdot a \ldots \cdot a$. We write $\sum_{i=1}^{n} p_i$ (where $p_i \in Terms(ACP)$) as a shorthand for $p_1 + p_2 \ldots + p_n$. As a special case, we let $\sum_{i=1}^{0} p_i$ denote δ. □

Theorem 9.3. There are (finite) A, γ, Δ such that there is **no** simulator $Sim = (C, B, R)$ for language *Terms*(ACP) that *respects* (is sound, complete and compatible with) $LTS/\underline{\leftrightarrow}$. □

Proof. In [BK83, Tau89] it is shown that we can choose finite A, γ, Δ in such way that we can exhibit a term $\mathrm{U2CM_n} \in Terms(\mathrm{ACP})$ (for each $n \in \mathbb{N}$) whose LTS behaves like a universal 2-counter machine on input n. Then $LTS/\underline{\leftrightarrow} \models \mathrm{U2CM_n} = X$ (where $X \stackrel{\mathrm{def}}{=} a \cdot X \in \Delta$) iff the counter machine diverges. This is a nonrecursive problem [HU79]: let K be a recursively enumerable but not recursive subset of $\mathbb{N}$, then $n \notin K \iff LTS/\underline{\leftrightarrow} \models \mathrm{U2CM_n} = X$.

Now suppose $Sim = (C, B, R)$ is a simulator of the intended kind and let $B(C(X), a) = k$ with $k \geq 1$. And define the process $Y \equiv \sum_{i=1}^{k+1} \mathrm{U2CM_n} \cdot b^i$ by using notation 9.2. Then we have:

- $n \in K \Rightarrow B(C(Y), a) \geq k + 1$, because the initial state $C(Y)$ of process Y has at least $k + 1$ outgoing a-edges by *compatibility* with $LTS/\underline{\leftrightarrow}$.
- $n \notin K \Rightarrow B(C(Y), a) = B(C(X), a) = k$ by *completeness*.

Combining the last two implications, we have $B(C(Y), a) = k \iff n \notin K$ which establishes that B can not be computable. □

We conjecture that we can prove similar (negative) results as theorem 9.3 in a setting of $LTS/\!\sim_{FT}$ (failure trace), $LTS/\!\sim_{R}$ (ready), $LTS/\!\sim_{F}$ (failure), $LTS/\!\sim_{CT}$ (completed trace).[3] However, there **do** exist simulators respecting $LTS/\!\sim_{T}$ (trace) and $LTS/\!\sim_{RT}$ (ready trace) as will be shown in the next section.

10 The Compatibility of a Trace Simulator

In this section we show that there exist simulators which respect trace semantics. For technical conveniance we shall use an LTS model which incorporates an empty process (ϵ) and therewith is slightly different from the LTS model given in definition 3.5. The empty process can be added to the syntax of ACP by the following production rule:

[3] The definition of these equivalences can be found in [Gla90].

$p := \ldots \mid \epsilon$.

The terms of ACP extended with ϵ is denoted by $Terms(\mathrm{ACP}_\epsilon)$.

Definition 10.1. *(Another LTS model for ACP.)* Let $\longrightarrow$ be the transition relation defined by the SOS rules given in table 3. Then, we define the LTS: $SOS_\epsilon(p) := (Terms(\mathrm{ACP}_\epsilon), A_\surd, \longrightarrow, p)$ as the interpretation of a term $p \in Terms(\mathrm{ACP})$ in the *LTS* model. □

ϵ	:	$\epsilon \xrightarrow{\surd} \top$
$a \in A$	:	$a \xrightarrow{a} \epsilon$
$\cdot$	:	$\dfrac{p \xrightarrow{a} p'}{p \cdot q \xrightarrow{a} p' \cdot q}$ $\qquad$ $\dfrac{p \xrightarrow{\surd} \top \quad q \xrightarrow{u} q'}{p \cdot q \xrightarrow{u} q'}$
$+$	:	$\dfrac{p \xrightarrow{u} p'}{p + q \xrightarrow{u} p' \quad q + p \xrightarrow{u} p'}$
$\|\|$	:	$\dfrac{p \xrightarrow{a} p'}{p \,\|\| q \xrightarrow{a} p' \,\|\| q \quad q \,\|\| p \xrightarrow{a} q \,\|\| p'}$
		$\dfrac{p \xrightarrow{\surd} \top \quad q \xrightarrow{\surd} \top}{p \,\|\| q \xrightarrow{\surd} \top}$
		$\dfrac{p \xrightarrow{a} p' \quad q \xrightarrow{b} q'}{p \,\|\| q \xrightarrow{c} p' \,\|\| q'}\ \gamma(a,b) = c$
∂_H	:	$\dfrac{p \xrightarrow{u} p' \quad u \notin H}{\partial_H(p) \xrightarrow{u} \partial_H(p')}$
Recursion	:	$\dfrac{p \xrightarrow{u} p' \quad (X \stackrel{\mathrm{def}}{=} p) \in \Delta}{X \xrightarrow{u} p'}$

Table 3: SOS rules for $Terms(\mathrm{ACP}_\epsilon)$.

Definition 10.2. *(Trace equivalence.)* The *trace set* of an LTS: $g = (S, L, \rightarrow, s)$ is given by the set

$$Tr(g) := \{w \mid s \xrightarrow{w \equiv a^1 \ldots a^n}\}.$$

Two LTS's $g, h \in LTS$ are *trace equivalent,* notation $g \sim_T h$ iff $Tr(g) = Tr(h)$. □

Definition 10.3. *(A Trace Model for ACP.)* Define the equivalence class:

$$\{g \in LTS \mid g \sim_T SOS_\epsilon(p)\}$$

as the interpretation of a term $p \in Terms(\mathrm{ACP})$ in the trace model $LTS/\sim_T$. □

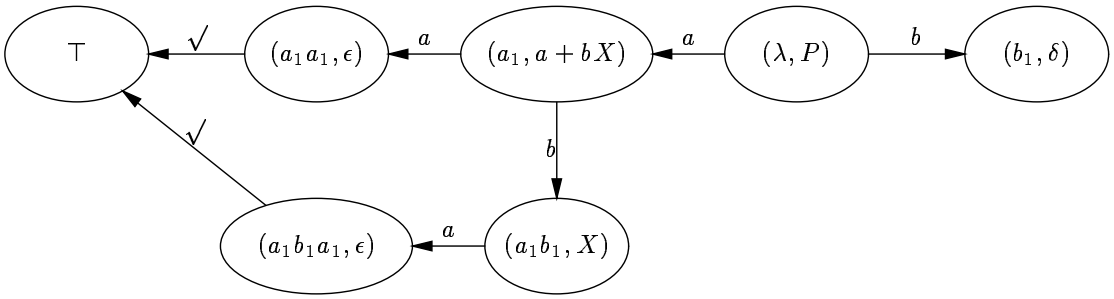

Figure 3: $LTS_{Sim}(P)$ where $P \stackrel{\text{def}}{=} abX + aa + b\delta$, $X \stackrel{\text{def}}{=} a$.

Notation 10.4. Let $\{p_0, \ldots, p_n\}$ be a set of ACP terms. We write $\sum_{p \in \{p_0, \ldots, p_n\}} p$ as a shorthand for $p_0 + \ldots + p_n$. As a special case, we let $\sum_{p \in \emptyset} p$ stand for δ. □

Theorem 10.5. There exist a simulator *respecting* the trace model $LTS/\sim_T$ as given in definition 10.3. □

Proof. Define a simulator Sim for $Terms(\text{ACP})$ as follows.

- Let $\mathsf{S} = Terms(\text{ACP}_\epsilon)$ and $\text{Act} = A \cup \{\surd\}$.
- $C(p) = p$ for all $p \in \mathsf{L}$.
- Let $Succ(p, u) = \{p' \mid p \xrightarrow{u} p'\}$ be the u-successor set of $p \in \mathsf{S}$, where $\longrightarrow$ is defined by the action rules in table 3.
  - $B(p, u) = \begin{pmatrix} 0 & \text{if } p \not\xrightarrow{u} \\ 1 & \text{if } p \xrightarrow{u} \end{pmatrix}$.
  - $R(p, u, i) = \begin{pmatrix} \sum_{p' \in Succ(p,u)} p' & \text{if } i = 1 \\ \bot & \text{otherwise} \end{pmatrix}$. Note how the residue function is defined by the Σ notation (notation 10.4). The Σ notation is used for taking the alternative composition of all a-successor states of p. This is the way how this simulator determinises the next state.

To guide the intuition of this simulator, the simulation tree of term $P \stackrel{\text{def}}{=} abX + aa + b\delta$ (with $X \stackrel{\text{def}}{=} a$) is displayed in figure 3. □

The proof of the following theorem shows that things get more involved when we do not have an empty process at our service.

Theorem 10.6. There exist a simulator *respecting* the trace model $LTS/\sim_T$ as given in definition 3.6. □

Proof. Define a simulator $Sim = (C, B, R)$ for $\mathsf{L} = Terms(\text{ACP})$ as follows.

- Let $\mathsf{S} = Terms(\text{ACP})$ and let $\text{Act} = A$.
- $C(p) = p$ for all $p \in \mathsf{L}$.
- Let $\longrightarrow$ be the transition relation defined by the action rules of table 2.

- $B(s,a) = \left(\begin{array}{ll} 0 & \text{if } s \stackrel{a}{\not\longrightarrow} \\ 2 & \text{if } (s \stackrel{a}{\longrightarrow} s' \not\equiv \top) \wedge s \stackrel{a}{\longrightarrow} \top \\ 1 & \text{otherwise} \end{array} \right)$.
- Let $Succ(p,a) = \{p' \mid p \stackrel{a}{\longrightarrow} p'\}$ be the a-successor set of $p \in \mathsf{S}$.
  - $R(s,a,1) = \left(\begin{array}{ll} \top & \text{if } Succ(s,a) = \{\top\} \\ \sum_{s' \in Succ(s,a)} s' & \text{otherwise} \end{array} \right)$
  - $R(s,a,2) = \left(\begin{array}{ll} \top & \text{if } (s \stackrel{a}{\longrightarrow} s' \not\equiv \top) \wedge s \stackrel{a}{\longrightarrow} \top \\ \bot & \text{otherwise} \end{array} \right)$
  - $R(s,a,i) = \bot$ if $i > 2$.

□

There also exists a simulator respecting ready trace semantics.

Definition 10.7. *(Ready Trace.)* The *ready trace set* of an LTS: $g = (S, L, \rightarrow, s)$ is given by

$$\textit{ready-trace}(g) = \{A_0 a_1 A_1 \ldots a_n A_n \mid$$
$$\exists s_0, \ldots, s_n : s = s_0 \stackrel{a_1}{\longrightarrow} s_1 \cdots \stackrel{a_n}{\longrightarrow} s_n,\ s_i \stackrel{a}{\longrightarrow} \iff a \in A_i, 0 \leq i \leq n\}.$$

Two LTS's $g, h \in LTS$ are *ready trace equivalent*, written $g \sim_{RT} h$, iff *ready-trace*(g) = *ready-trace*(h). □

Theorem 10.8. There exists a simulator *Sim* for language *Terms*(ACP) *respecting* $LTS/\sim_{RT}$ (ready trace semantics) as given in definition 3.6. □

Proof. Very technical and omitted. □

We expect that simulators respecting $LTS/\sim_T$ or $LTS/\sim_{RT}$ are in theory computationally intractable as they are always based on some notion of transition determinisation. From automata theory, we know that in general 'determinisation algorithms' are PSPACE-complete (see [HU79]). But in [HU79] it is also claimed that these determinisation algorithms only use exponential space in rare circumstances. This suggests that the preformence of trace based simulators may be not that bad in practice.

11 The Semantics of an Efficient Simulator

The only practical (polynomial) simulator we encountered in the previous sections was the SOS simulator (simulator 5.1). However, we could not find a reasonable model with respect to this simulator is sound and complete. So the existence of a reasonable semantic characterisation of simulator 5.1 is still an open problem.

In this section, we show there exists a polynomial simulator that is sound and complete with respect to an equational term model that is strictly weaker than bisimulation and graph isomorphism.

In table 4, we can find an equational system ACP^- that is a weakened version of the standard equations of ACP (table 5).

The equations in Table 5 and Table 4 contain two auxiliary operators $\between$ *(left-merge)* and $\mid$ *(communication-merge)* for axiomatising the $\|$-operator. $p \between q$ is $p \parallel q$ but with the restriction that the first step comes from p, and $p \mid q$ is $p \parallel q$ but with a synchronisation action as the first step. The terms of ACP extended with $\between$ and $\mid$ is denoted by *Terms*(ACP, AUX). For handling recursively declared

$A1^-$	$ax + by = by + ax \quad \text{if } a \not\equiv b$
	$ax + b = b + ax \quad \text{if } a \not\equiv b$
	$a + by = by + x \quad \text{if } a \not\equiv b$
	$a + y = y + x \quad \text{if } a \not\equiv b$
A2	$(x + y) + z = x + (y + z)$
A4	$(x + y)z = xz + yz$
A5	$(xy)z = x(yz)$
$A6^+$	$x + \delta = x$
	$\delta + x = x$
A7	$\delta x = \delta$
CF1	$a \mid b = \gamma(a, b) \quad \text{if } \gamma(a, b) \downarrow$
CF2	$a \mid b = \delta \qquad \text{otherwise}$
CM1	$x \parallel y = x \between y + y \between x + x \mid y$
CM2	$a \between x = ax$
CM3	$ax \between y = a(x \parallel y)$
CM4	$(x + y) \between z = x \between z + y \between z$
CM5	$ax \mid b = (a \mid b)x$
CM6	$a \mid bx = (a \mid b)x$
CM7	$ax \mid by = (a \mid b)(x \parallel y)$
CM8	$(x + y) \mid z = x \mid z + y \mid z$
CM9	$x \mid (y + z) = x \mid y + x \mid z$
D1	$\partial_H(a) = a \quad \text{if } a \notin H$
D2	$\partial_H(a) = \delta \quad \text{if } a \in H$
D3	$\partial_H(x + y) = \partial_H(x) + \partial_H(y)$
D4	$\partial_H(x \cdot y) = \partial_H(x) \cdot \partial_H(y)$

Table 4: Module ACP^-

A1	$x + y = y + x$
A2	$(x + y) + z = x + (y + z)$
A3	$x + x = x$
A4	$(x + y)z = xz + yz$
A5	$(xy)z = x(yz)$
A6	$x + \delta = x$
A7	$\delta x = \delta$
CF1	$a \mid b = \gamma(a, b)$ if $\gamma(a, b) \downarrow$
CF2	$a \mid b = \delta$ otherwise
CM1	$x \parallel y = x \between y + y \between x + x \mid y$
CM2	$a \between x = ax$
CM3	$ax \between y = a(x \parallel y)$
CM4	$(x + y) \between z = x \between z + y \between z$
CM5	$ax \mid b = (a \mid b)x$
CM6	$a \mid bx = (a \mid b)x$
CM7	$ax \mid by = (a \mid b)(x \parallel y)$
CM8	$(x + y) \mid z = x \mid z + y \mid z$
CM9	$x \mid (y + z) = x \mid y + x \mid z$
D1	$\partial_H(a) = a$ if $a \notin H$
D2	$\partial_H(a) = \delta$ if $a \in H$
D3	$\partial_H(x + y) = \partial_H(x) + \partial_H(y)$
D4	$\partial_H(x \cdot y) = \partial_H(x) \cdot \partial_H(y)$

Table 5: Module ACP

REC	$X = p$ if $(X \stackrel{\text{def}}{=} p) \in \Delta$

Table 6: Module REC.

PR1	$\pi_n(a) = a$
PR2	$\pi_1(ax) = a$
PR3	$\pi_{n+1}(ax) = a \cdot \pi_n(x)$
PR4	$\pi_n(x + y) = \pi_n(x) + \pi_n(y)$

Table 7: Module PR.

AIP	$\dfrac{\pi_n(x) = \pi_n(y) \ \text{ for all } n \geq 1}{x = y}$

Table 8: Module AIP.

ACP processes we use the modules REC (Table 6), PR (Table 7) and AIP (Table 8). Modules ACP and ACP^- extended with REC, PR and AIP are denoted by $\mathrm{ACP}_\#$ and $\mathrm{ACP}^-_\#$ respectively.
We let $\equiv_{\mathrm{ACP}^-_\#}$ be the equivalence relation induced by module $\mathrm{ACP}^-_\#$:

$$p \equiv_{\mathrm{ACP}^-_\#} q \Longleftrightarrow \mathrm{ACP}^-_\# \vdash p = q.$$

Theorem 11.1. There exists a *polynomial* simulator that is *sound* and *complete* with respect to $Terms/\equiv_{\mathrm{ACP}^-_\#}$. □

For proving this theorem we introduce the following definition.

Definition 11.2. *(Ordered head normal form.)* A term $p \in Terms(\mathrm{ACP}, \mathrm{AUX})$ is in *ordered head form* if p is of the form $\hat{p} \equiv \sum_{a \in B} \sum_{i=1}^{n_a} a p_a^i$ or $\hat{p} \equiv \sum_{a \in B} \sum_{i=1}^{n_a} a$ where $B \subseteq A, n_a > 0, p_a^i \in Terms(\mathrm{ACP}, \mathrm{AUX})$.
Actions $a \in B$ are called *head actions* and term p_a^i is the *residue term* of the i^{th} occurrence (from left to right conform notations 9.2, 10.4) of head action a in the ordered head form of p. □

For example, terms $aa + aa + ba$ and $ba + aa + aa$ are in ordered head form. Term $aa + ba + aa$ is not in ordered head form.

Proof of theorem 11.1. Define $Sim = (C, B, R)$ for $Terms(\mathrm{ACP})$ as follows.

- Let $\mathsf{Act} = A$ and let $\mathsf{S} = Terms(\mathrm{ACP}, \mathrm{AUX})$.
- $C(p) = p$.
- It is easy to see that with the equations of ACP^- and REC we can rewrite a term p into ordered head form $\hat{p} \equiv \sum_{a \in B} \sum_{i=1}^{n_a} a p_a^i$ or $\hat{p} \equiv \sum_{a \in B} \sum_{i=1}^{n_a} a$ in polynomial time. Then define:
  - $B(p, a) = n_a$.
  - $R(p, a, i) = \begin{pmatrix} \top & \text{if } \hat{p} \equiv \sum_{a \in B} \sum_{i=1}^{n_a} a \\ p_a^i & \text{if } \hat{p} \equiv \sum_{a \in B} \sum_{i=1}^{n_a} a p_a^i \end{pmatrix}$.

The intuition of the simulator is as follows. Given an action a and a term p, the branching function B computes the number of times action a appears as head action in the ordered head form of p. And given an action a, a term p and a number $i > 0$, the residue function R returns the residue term of the i^{th} occurrence (from left to right) of action a in the ordered head form of term p.

The simulation tree of term $P \stackrel{\mathrm{def}}{=} abX + aa + b\delta$ (with $X \stackrel{\mathrm{def}}{=} a$) is displayed is given in figure 4. □

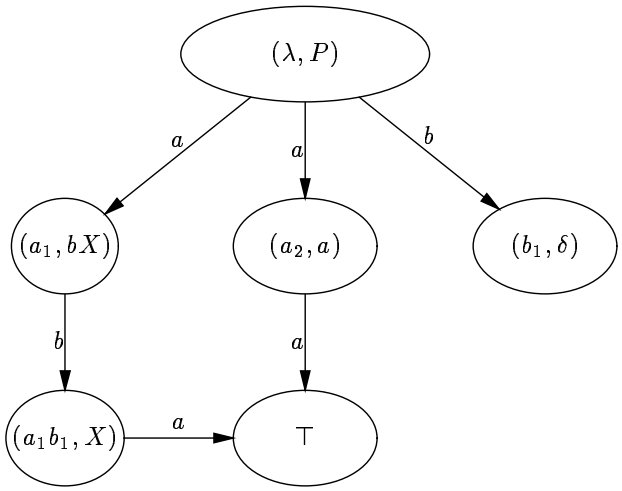

Figure 4: $LTS_{Sim}(P)$ where $P \stackrel{\text{def}}{=} abX + aa + b\delta$, $X \stackrel{\text{def}}{=} a$.

12 Summary and Conclusion

The table below summerises the results of this paper; it shows the existence or *non*-existence of simulators respecting (being sound, complete and compatible with respect to) the well-known process models that have emerged from process theory [Gla90]. In addition, information is given about the computational complexity of a simulator in question: 'P' indicates the existence of a polynomial simulator and '¬P' indicates that we could not find a polynomial simulator with the required properties. A question mark (?) in the table expresses that the existence or the *non*-existence of a certain simulator is conjectured (Yes?, No?). The symbols I, B, RT, FT, R, F, CT, T are taken from [Gla90] and resp. stand for the models $LTS/\simeq$ (graph isomorphism), $LTS/\leftrightarrows$ (bisimulation), $LTS/\sim_{RT}$ (ready trace), $LTS/\sim_{FT}$ (failure trace), $LTS/\sim_R$ (ready), $LTS/\sim_F$ (failure), $LTS/\sim_{CT}$ (completed trace), $LTS/\sim_T$ (trace).

	I	B	RT	FT	R	F	CT	T
sound	Yes P	Yes P	Yes P	Yes P	Yes P	Yes P	Yes? P	Yes P
sound & complete	Yes ¬P	Yes ¬P	Yes ¬P	Yes? ¬P	Yes? ¬P	Yes? ¬P	Yes ¬P	Yes ¬P
sound & complete & compatible	Yes ¬P	No ¬P	Yes ¬P	No? ¬P	No? ¬P	No? ¬P	No? ¬P	Yes ¬P

One can make up from the table above there do not exist simulators respecting B as already claimed by theorem 9.3. In contrast with this, it is displayed in the first column that there does exists a simulator respecting graph isomorphism. However, we do not expect that simulators respecting graph isomorphism are computationally tractable as indicated by '¬P' at the bottom of the first column. Furthermore one can read from the table that it is conjectured that there are *no* simulators respecting FT, R, F and CT. At last, it is conjectured that there do exist simulators respecting RT and T.

Although it is very likely that in theory these simulators are computationally intractable, they may perform well in practise as already claimed in section 9.

The first row of the table says there are simulators that are sound with graph isomorphism and thus with all the other models. For instance by proposition 8.3 we know that simulator 5.1 is sound with respect to all the models in the table. All these observations from the table presume that the equivalence induced by a practical simulator must be less identifying than graph isomorphism. As an illustration of this presumption, we presented an efficient (polynomial) simulator which is less identifying than graph isomorphism in the previous section.

References

[BK83] J.A. Bergstra and J.W. Klop. The algebra of recursively defined processes and the algebra of regular processes. Report IW 235, CWI, Amsterdam, 1983.

[BW90] J.C.M. Baeten and W.P. Weijland. *Process Algebra.* Cambridge Tracts in Theoretical Computer Science 18. Cambridge University Press, 1990.

[CPS90] R. Cleaveland, J. Parrow, and B. Steffen. A semantics-based tool for the verification of finite-state systems. In *Proceedings of the Ninth IFIP Symposium on Protocol Specification, Testing and Verification*, Lecture Notes in Computer Science, pages 287–302. North-Holland, 1990.

[Eer91] H. Eertink. SMILE detailed design document. LOTOSPHERE technical report Lo/WP2/T2.2/UT/N0012, University of Twente, 1991.

[Gla90] R.J. van Glabbeek. The linear time – branching time spectrum. In J.C.M. Baeten and J.W. Klop, editors, *Proceedings CONCUR 90,* Amsterdam, volume 458 of *Lecture Notes in Computer Science*, pages 278–297. Springer-Verlag, 1990.

[HU79] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[Plo81] G.D. Plotkin. A structural approach to operational semantics. Report DAIMI FN-19, Computer Science Department, Aarhus University, 1981.

[Rog67] H. Rogers. *Theory of Recursive Functions and Effective Computability.* McGraw-Hill Book Co., 1967.

[SPE92] SPECS. Intermediate report on methods and tools in CRL/MR. Document D5.19, European project RACE 1046: Specification Environment for Communication Software (SPECS), 1992.

[Tau89] D.A. Taubner. *Finite representation of CCS and TCSP programs by automata and Petri nets*, volume 369 of *Lecture Notes in Computer Science.* Springer-Verlag, 1989.

[Tre89] J. Tretmans. HIPPO: a LOTOS simulator. In van Eijk et al. [vEVD89], pages 391–396.

[vE89] P.H.J. van Eijk. The design of a simulator tool. In van Eijk et al. [vEVD89], pages 351–390.

[Vel93] G.J. Veltink. The PSF toolkit. In *Computer Networks and ISDN Systems*, number 25. North-Holland, 1993.

[Ver92] J.A. Verschuren. A simulator for μCRL in ASF+SDF. Report P9203, Programming Research Group, University of Amsterdam, 1992.

[vEVD89] P.H.J. van Eijk, C.A. Vissers, and M. Diaz, editors. *The Formal Description Technique LOTOS.* North-Holland, Amsterdam, 1989.