A Theory for Simulators

HENRI KORVER

CWI, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands

This paper presents a formalization of the notion simulator **for process languages like ACP, CCS, (μ)CRL, LOTOS and PSF. Precise definitions for the notions** simulator **and** simulation **are given. Moreover, the equivalence that a simulator induces on the explored process terms is investigated. This is done by considering two processes 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.**

1. INTRODUCTION

Nowadays the so-called simulator tools can not be thought away in the validation of concurrent system specifications. A number 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 state space of process term in a certain language. Examples are the simulators that have been built for CCS [4], (μ)CRL [14, 18], LOTOS [5, 6, 16, 7] and PSF [17]. The basic working 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.

To our knowledge this idea of the working of a simulator has never been formalized in a general and structural way. In this paper, we propose an exact definition for the notion simulator. As far as we are aware all existing simulator tools can be described as an instance of this definition. The motivation of this work is that it allows for studying simulator tools in a precise and meaningful way.

More precisely, we define a simulator as a triple $Sim = (C, M, R)$ of recursive (computable) functions. From a process term that is fed into the simulator, the *conversion* function C computes the initial state of its exploration. The *menu* function M 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.

Once we have formalized the notion of a simulator, we can investigate the equivalence it induces on process terms. In particular, two process terms p and q are equivalent exactly when each simulation of p is also a simulation of q and vice versa. A simulation of a term p is defined as a *finite* alternating sequence of menus offered by the simulator and choices from these menus (see Definition 3.1). To guide the intuition, consider the two CCS processes $a.a.0+a.a.0$ and $a.a.0+a.(a.0+a.0)$. Although they are obviously bisimilar, for example the Concurrency Workbench [4] (a toolbox including a simulator for CCS) already distinguishes them in the first menus:

$$M(a.a.0 + a.a.0) = \{1 : \xrightarrow{a} a.0\}$$

and $M(a.a.0 + a.(a.0 + a.0)) =$

$$\{1 : \xrightarrow{a} a.0;\ 2 : \xrightarrow{a} a.0 + a.0\}.$$

This poses the question: do there exist (in theory) simulators inducing bisimulation equivalence? If so, then we may be able to develop simulator tools that do not confuse us with too many details and that still visualise all the aspects of processes one could be interested in.* If not, then we know that a simulator does not respect a fundamental process equivalence.

In this paper, it is shown that there *does* exist a (highly theoretical) simulator inducing bisimulation equivalence (see Theorem 7.1). However the working of this simulator goes beyond any reasonable intuition. For instance, the generated simulations do not match with our 'graph interpretation' of processes. For ruling out such 'unreasonable' simulators we introduce the *compatibility* restriction in Definition 8.1. A simulator is compatible with a particular equivalence class model† of process graphs if the graph of all the possible simulations of a process term p is contained in the equivalence class interpretation of p. For example, a simulator computing the menu

$$M(a.0 + a.a.0) = \{1 : \xrightarrow{a} a.0\}$$

is not compatible with the standard bisimulation model [11] where there are always two a-edges in the graph interpretation of $a.0 + a.a.0$.

With the compatibility restriction we prove in Section 8 that in general there do *not* exist simulators inducing bisimulation equivalence. Furthermore, there are

*Following the paradigm that bisimulation equivalence is the most concrete behavioural equivalence one wants to impose, e.g. [1].

†For technical reasons we restrict ourselves to process models that can be represented by an equivalence class model of labelled transition systems (LTS) denoted by $LTS/\sim$. Nearly all existing process models can be represented in such a way.

simulators inducing coarser equivalences like ready and trace equivalence. Unfortunately, such simulators are unlikely to run in polynomial time (at least we could not find them). We had the same experience with graph isomorphism. Concluding, we expect that the equivalence induced by an efficient simulator is at least finer (less identifying) than bisimulation equivalence.

2. PRELIMINARIES

2.1. 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 2.1 *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 relation where* $\top$ *is a distinguished element called the* termination *state;*
- $r \in S$ *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: (1) $s \xrightarrow{a} s'$ for $(s, a, s') \in \longrightarrow$ (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) $s \xrightarrow{a_1 \cdots a_n} s_n$ for $s \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \cdots \xrightarrow{a_n} s_n$. A state s is called a *deadlock* state if $s \not\equiv \top$ and $s \not\rightarrow$.

Definition 2.2 *(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 2.3 *(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* $s' R t'$.
  – *if* sRt *and* $t \xrightarrow{a}_2 t'$, *then there is a* $s' \in S_1$ *with* $s \xrightarrow{a}_1 s'$ *and* $s' R t'$.
- – *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.*

2.2. A process specification language

As a running example, we here present the language of ACP [3]. ACP is chosen here as one of the many algebraic formalisms for specifying processes (LTS's), like CCS, CSP and MEIJ E, but of course the other candidates can be used equally well.

We consider the following ACP syntax:

$$p := \delta \mid p \cdot p \mid p + p \mid p \parallel p \mid a \mid \partial_H(p) \mid X \quad \text{where}$$

- a ranges over a *finite* set of actions A.
- $H \subseteq A$.
- X ranges over a *finite* set $Var = \{X, Y, \ldots\}$ of variables. We say that a process expression p is *guarded* iff every variable occurrence in p occurs in a subexpression aq of p. Recursive processes are defined by a finite set of guarded equations

  $$\Delta = \{X_i \stackrel{\text{def}}{=} p_i \mid 1 \leq i \leq k\}$$

  where the X_i are distinct variables from Var, and the p_i are guarded ACP expressions with free variables in $Var(\Delta) = \{X_1, \ldots, X_k\}$.

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 build process terms are sequential composition $p \cdot p$, summation $p + p$, parallel composition $p \parallel p$ and encapsulation $\partial_H(p)$ 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 2.4 *(The LTS specified by an ACP term.) Suppose that an action set* A, *a* symmetric *communication function* $\gamma : A \times A \rightarrow A$ *and a (recursive) process declaration* Δ *are given. Then let* $\longrightarrow$ *be the transition relation defined by the action rules given in Table 1. 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 1 follow the structure of the process terms and are therefore called SOS (acronym for Structural Operational Semantics) rules [12].

Definition 2.5 *(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* $LTS/\sim$.

3. A DEFINITION FOR A SIMULATOR

In the definition below we formalize a simulator as a 3-tuple $Sim = (C, M, R)$ consisting of a conversion-, menu- and a residue function.

TABLE 1. SOS rules for *Terms*(ACP).

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

Definition 3.1 *(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, let* $\mathsf{S} = \{s^3, s^4, \ldots\}$ *be an enumerable set of states and let* $\mathsf{S}' = \{s^1, s^2, s^3, s^4, \ldots\}$ *be the enumerable set* S *including the* special *states* $s^1 \equiv \bot$ *and* $s^2 \equiv \top$. *The function triple* $Sim = (C, M, R)$ *where*

$C : \mathsf{L} \to \mathsf{S}$ *(Conversion function)*
$M : \mathsf{S} \to \mathcal{P}_{fin}(\mathsf{Act} \times \mathbb{N})$ *(Menu function)*
$R : \mathsf{S} \times \mathsf{Act} \times \mathbb{N} \to \mathsf{S}'$ *(Residue function)*

is a simulator *(for language* L*) if*

- C, M *and* R *are* computable *functions with respect to the enumerations of* L, Act, S *and* S'.‡
- $(a,i) \notin M(s) \iff R(s,a,i) = \bot$ *for all* $s \in \mathsf{S}$, $a \in \mathsf{Act}$ *and* $i \in \mathbb{N}$.

In the definition above, $\mathcal{P}_{fin}(\mathsf{Act} \times \mathbb{N})$ denotes the set of all finite subsets of $\mathsf{Act} \times \mathbb{N}$.

‡Formally this means that C, M and R can be coded by the *recursive* functions $c, m : \mathbb{N} \to \mathbb{N}$ and $r : \mathbb{N}^3 \to \mathbb{N}$ defined by $c(t) = k \iff C(p^t) = s^k$; $m(k) = CI(\{\langle l, i\rangle \mid (a^l, i) \in M(s^k)\})$; and $r(k,l,i) = s^u \iff R(s^k, a^l, i) = s^u$. The canonical index *(CI)* of $\emptyset$ is 0 and of $\{k_1, k_2, \ldots, k_l\}$ it is the number $2^{k_1} + 2^{k_2} + \ldots + 2^{k_l}$. An ordered pair $\langle k, l\rangle$ is coded by $\frac{1}{2}(k^2 + 2kl + l^2 + 3k + l)$, see [13]. However readers who are not familiar with recursion theory can skip these definitions without any problem.

The intuition of the functions C, M, 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 menu function M computes the set of actions that the simulated process can perform in state s. These actions are linked with a natural number $i > 0$ to distinguish actions that are the same but lead to different states.

Given a choice from the menu $(a,i) \in M(s)$ in the current state s, the residue function R computes the next state. In other words, $(s, a, R(s,a,i))$ is the a-transition that is explored by the simulator by choosing (a,i) from the menu $M(s)$. $M(s) = \{a_1, a_2,\ dotsa_n\}$ and The range of R includes two special symbols $\bot$ and $\top$. $R(s,a,i) = \top$ means that after choosing (a,i) from the menu of s the termination state is reached. When this is the case the simulator stops the exploration. The $\bot$ symbol is used to express that the next state is undefined. E.g. $R(s,a,i) = \bot$ when (a,i) is not a choice from the menu of s (see the last line of Definition 3.1).

Definition 3.2 *(Polynomial simulator.) A simulator* $Sim = (C, M, R)$ *is* polynomial *if the functions* C, M *and* R *can be computed in polynomial time with respect to the size of their arguments.*

4. AN EXAMPLE OF A SIMULATOR

Many simulator tools [4, 5, 6, 16, 14] for process languages are implemented via action (SOS) rules. As an example, we instantiate a simulator for ACP via the SOS rules given in Table 1. This example is a simplification but gives the underlying idea of the working of many existing simulator tools.

Example 4.1 (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 $Sim = (C, M, R)$ 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 1.
  - $M(p) = \{(a, i) \mid 0 < i \leq |Succ(p, a)|\}$
  - The residue $R(p, a, i)$ is defined as the i^{th} element in the set $Succ(p, a)$ with respect to an arbitrary but fixed *strict*[§] ordering $\prec$ on $\mathsf{S} \cup \{\top\}$. Formally, $R(p, a, i) =$

$$\begin{cases} q & \text{if } q \in Succ(p,a); \text{ and} \\ & \forall j \in \{0, \ldots, i-1\} : R(p,a,j) \prec q; \text{ and} \\ & \forall j \in \{i+1, \ldots, |Succ(p,a)|\} : q \prec R(p,a,j) \\ \bot & \text{otherwise} \end{cases}$$

  In the example below, we assume that $\prec$ satisfies: $a \prec b \prec X \prec bX$.

For compact notation we often write a_i for (a, i).

Now let us try to understand the working of the simulator given in Example 4.1. 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 the simulator. 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 $a \prec bX$ (see the definition above). 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 the simulator given in Example 4.1 can be computed in polynomial time.

Proposition 4.1 The simulator given in Example 4.1 is *polynomial*.

There are also simulators that are based on rewriting process terms into head normal form, e.g. the simulator for μCRL developed by Emile Verschuren [18] and the PSF simulator [17]. In [10] it is shown how such simulators can also be defined in our framework.

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

5. DEFINING A SESSION WITH A SIMULATOR

A session of a term p with a simulator $Sim = (C, M, R)$ 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 5.1 *(A session with a simulator.) Let $Sim = (C, M, R)$ be a simulator for L.*

- *$M(C(p))$ is a* session *(for $p \in \mathsf{L}$).*
- *Let φ be a* session *ending in $M(s)$ and let $(a, i) \in M(s)$. If $R(s, a, i) = \top$ then $\varphi(a, i)$ is a* session *called a* termination session*; otherwise $\varphi(a, i)M(R(s, a, i))$ is a* session*. In case $M(s) = \emptyset$ we call φ a* deadlock session*, and in case $M(s) \neq \emptyset$ we call φ a* partial session*.*

Example 5.1 Below all the sessions of $P \stackrel{\text{def}}{=} abX + aa + b\delta$, $X \stackrel{\text{def}}{=} a$ with the simulator given in Example 4.1 are summerized.

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, 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 into a deadlock state.

6. THE SESSION SET SEMANTICS OF A SIMULATOR

In this section, we provide a simulator with a semantics. The session set semantics is in our opinion the most logical first step to provide a simulator with a semantics. In this semantics, a process term is modeled by the set containing all possible sessions that can be obtained by exploring this term with the simulator.

From this session set model, we derive two useful models, the simulation set and simulation graph model, which both make the same identifications on processes as the session set model. The simulation set model is introduced because it is an efficient representation of the session set model and allows for easy notation. The simulation graph model is introduced as a handy representation for formulating a useful semantic criterion for simulators in Definition 8.1.

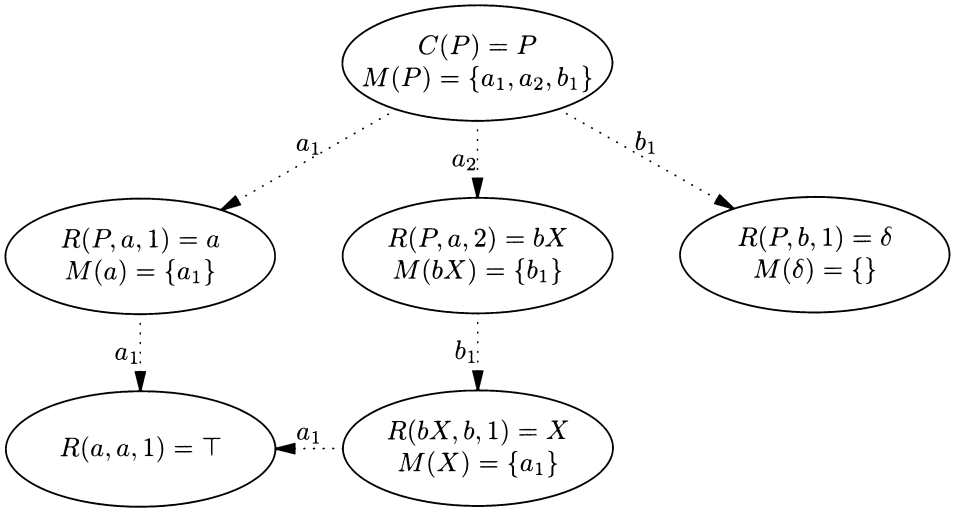

FIGURE 1. The working of the example simulator on the input of the process $P \stackrel{\text{def}}{=} abX + aa + b\delta$, $X \stackrel{\text{def}}{=} a$.

6.1. Session set and simulation set model

Definition 6.1 *(Session set of a term induced by a simulator.) Let* $Sim = (C, M, R)$ *be a simulator for* L. *The* session set *of a process term* $p \in \mathsf{L}$ induced *by Sim is given by*

$$SES_{Sim}(p) = \{\textit{all}\ \text{simulation sessions}\ \textit{of}\ p\ \textit{with}\ Sim\ \}$$

Definition 6.2 *(Equivalence on terms induced by a simulator.) A simulator* $Sim = (C, M, R)$ induces *an* equivalence $\equiv_{Sim}$ *on* L *as follows:*

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

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

- *If* $\varphi = M(s^0)$ *then*
$$\overline{\varphi} \stackrel{\text{def}}{=} \lambda.$$
- *If* $\varphi = M(s^0)(a^0, i_0)M(s^1)\ldots(a^{n-1}, i_{n-1})M(s^n)$ *then*
$$\overline{\varphi} \stackrel{\text{def}}{=} (a^0, i_0)\ldots(a^{n-1}, i_{n-1}).$$
- *If* $\varphi = M(s^0)(a^0, i_0)M(s^1)\ldots(a^{n-1}, i_{n-1})M(s^n)(a^n, i_n)$ *then*
$$\overline{\varphi} \stackrel{\text{def}}{=} (a^0, i_0)\ldots(a^n, i_n)\,\dagger\,.$$

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 6.4 *(Simulation set induced by a simulator.) Let* $Sim = (C, M, R)$ *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 5.1, 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 6.1 *Let* $Sim = (C, M, R)$ *be a simulator for* L. *For every* $p, q \in \mathsf{L}$, *we have that the following two statements are equivalent:*

1. $SIM_{Sim}(p) = SIM_{Sim}(q)$
2. $SES_{Sim}(p) = SES_{Sim}(q)$.

6.2. Simulation graph model

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

Definition 6.5 *(The LTS of a term explored by a simulator.) The LTS of a term* $p \in \mathsf{L}$ explored *by a simulator* $Sim = (C, M, R)$ *is given by*

$$LTS_{Sim}(p) = ((\mathsf{Act} \times \mathbb{N})^* \times \mathsf{S}, \mathsf{Act}, \longrightarrow, (\lambda, C(p)))$$

where $\longrightarrow$ *is defined by the following rules (one for each* $a \in \mathsf{Act}, i \in \mathbb{N}, s \in \mathsf{S}, \sigma \in (\mathsf{Act} \times \mathbb{N})^*$*):*

$$(\sigma, s) \xrightarrow{a} \top \qquad \textit{if}\ R(s, a, i) = \top;$$

$$(\sigma, s) \xrightarrow{a} (\sigma a_i, R(s, a, i)) \qquad \textit{if}\ a_i \in M(s)\ \textit{and}\ R(s, a, i) \neq \top.$$

We often call $LTS_{Sim}(p)$ *the* simulation graph *of p.*

This definition shall be used to formalize a semantic criterion for simulators in Definition 8.1.

In Figure 2, one can see that a state, say q, in the simulation graph consists of two components $q \equiv (\sigma, s)$.

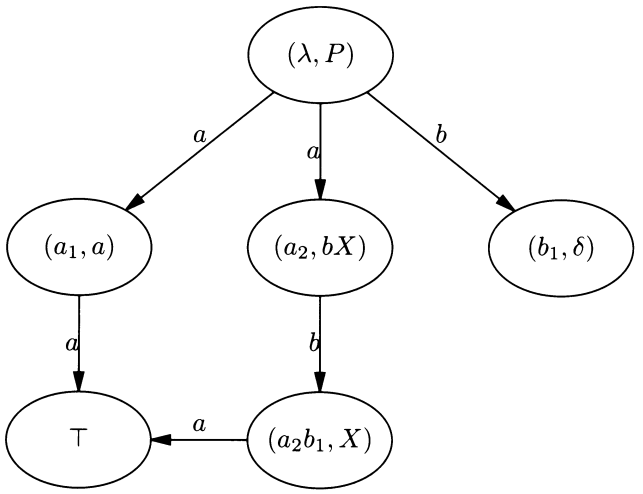

FIGURE 2. $LTS_{Sim}(P)$ where $P \stackrel{\text{def}}{=} abX + aa + b\delta$, $X \stackrel{\text{def}}{=} a$.

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 graph is built the second component can be left out.

The LTS of term P explored by the simulator given in Example 4.1 is presented in Figure 2.

COROLLARY 6.2 *Let $Sim = (C, M, R)$ be a simulator for L. For every $p, q \in \mathsf{L}$, the following four 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)*

When we write $LTS_{Sim}(p) = LTS_{Sim}(q)$ in the corollary above, we just mean that the graphs are *set* equivalent, e.g. $(\{1\}, \emptyset, \emptyset, 1) = (\{1, 1\}, \emptyset, \emptyset, 1)$. Note that this is less identifying than for instance isomorphism ($\simeq$). For example, $(\{1\}, \emptyset, \emptyset, 1) \simeq (\{2\}, \emptyset, \emptyset, 2)$ but $(\{1\}, \emptyset, \emptyset, 1) \neq (\{2\}, \emptyset, \emptyset, 2)$.

7. 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 [8]. 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 7.1 *(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 $p \equiv_{Sim} q \Rightarrow \mathcal{M} \models p = q$.*

[¶]A model $\mathcal{M} = (I, \mathcal{D})$ for L is a mapping $I : \mathsf{L} \to \mathcal{D}$ where $\mathcal{D}$ is a semantic domain. Let $p, q \in \mathsf{L}$, we write $\mathcal{M} \models p = q$ iff $I(p) = I(q)$.

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 7.2 *(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 given in Example 4.1 is related with the graph isomorphism model ($LTS/\simeq$) and the bisimulation model ($LTS/\leftrightarrow$).

Proposition 7.1 Let *Sim* be the simulator given in Example 4.1.

1. *Sim* is *sound* with respect to $LTS/\simeq$ as given in Definition 2.5.
2. *Sim* is **not** *complete* with respect to $LTS/\simeq$.
3. *Sim* is *sound* with respect to $LTS/\leftrightarrow$.
4. *Sim* is **not** *complete* with respect to $LTS/\leftrightarrow$.

Proof

1. Straightforward by inspection of the definitions of the simulator given in Example 4.1 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 the simulator given in Example 4.1 respects the ordering $a \prec b \prec X$.
3. Immediate by 7.1.1 by using the fact that isomorphism is strictly finer than bisimulation (in symbols: $\simeq \subseteq \leftrightarrow$).
4. Immediate by 7.1.2. Or as follows: $LTS/\leftrightarrow \models aa+aa = aa+a(a+a)$. But *Sim* distinguishes them: $M(aa+aa) = \{a_1\}$ and $M(aa+a(a+a)) = \{a_1, a_2\}$. ■

Proposition 7.1 says that the simulator given in Example 4.1 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 7.1 *There exists a simulator $Sim = (C, M, R)$ for $Terms(\mathrm{ACP})$ that is* sound *and* complete *with respect to $LTS/\leftrightarrow$ as given in Definition 2.5.*

We shall prove this theorem by constructing a simulator with the required properties. In doing this we need a projection operator which can be added to the syntax of ACP (as given in Section 2.2) as follows:

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

The terms of ACP extended with a projection operator is denoted by $Terms(\mathrm{ACP}, \mathrm{PR})$. 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 7.1.

LEMMA 7.2 *There exists a* computable *coding function* $\ulcorner . \urcorner : Terms(\mathrm{ACP}, \mathrm{PR}) \to \mathbb{N}$ *such that for all terms* $p, q \in Terms(\mathrm{ACP}, \mathrm{PR})$ *and all* $n, m \in \mathbb{N}$ *it holds that*

$$\ulcorner \pi_n(p) \urcorner = \ulcorner \pi_m(q) \urcorner \Longleftrightarrow LTS/\leftrightarrow \models \pi_n(p) = \pi_m(q).$$

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

Proof (Theorem 7.1.) Define a simulator $Sim = (C, M, R)$ for $Terms(\mathrm{ACP})$ as follows:

- Let $\mathsf{S} = Terms(\mathrm{ACP}) \times \mathbb{N}$ 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 $\pi_1(p)$.
- $M((p, n)) = \{(a, 1), (a, 2), \ldots, (a, \ulcorner \pi_n(p) \urcorner)\}$. So, the number of elements in the menu is exactly the encoding of the n^{th} projection of p. Note that the actual contents of this menu does not play a role but only its cardinality is what counts. By Lemma 7.2 we know that this coding takes care that bisimilar terms are identified.
- $R((p, n), a, i) =$

$$\begin{cases} (p, n+1) & \text{if } i \leq \ulcorner \pi_n(p) \urcorner \text{ and } \ulcorner \pi_n(p) \urcorner \neq \ulcorner \pi_{n+1}(p) \urcorner \\ \top & \text{if } \ulcorner \pi_n(p) \urcorner = \ulcorner \pi_{n+1}(p) \urcorner \\ \bot & \text{otherwise} \end{cases}$$

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 7.2. This encoding takes care that bisimilar terms are identified. It is obvious that the working of this simulator goes far beyond any reasonable intuition and in the next section we shall develop an criterion to rule out such simulator definitions.

As a final remark, we note that we could not find a bisimulation simulator which runs in polynomial time.

8. 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. For example the simulation graph of the ACP term $a + b$ induced by the simulator constructed in Section 7 only contains a-actions. But if we think in terms of for instance bisimulation semantics we find it reasonable that there must also occur an b action in the graph interpretation of $a + b$. 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 8.1 *(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 graph $LTS_{Sim}(p)$ is contained in the equivalence class interpretation of p in $LTS/\sim$ (denoted by $I(p)$).

REMARK 8.1 We write a^n for the n-fold sequential composition of an action $a \in A$ 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 8.1 *There are (finite)* A, γ, Δ *such that there is* **no** *simulator* $Sim = (C, M, R)$ *for language* $Terms(\mathrm{ACP})$ *that* respects *(is sound, complete and compatible with)* $LTS/\leftrightarrow$.

Proof In [2, 15] 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 $LTS/\leftrightarrow \models \mathrm{U2CM_n} = X$ (where $X \stackrel{\mathrm{def}}{=} a \cdot X \in \Delta$) iff the counter machine diverges. This is a nonrecursive problem [9]: let K be a recursively enumerable but not recursive subset of $\mathbb{N}$, then $n \notin K \Longleftrightarrow LTS/\leftrightarrow \models \mathrm{U2CM_n} = X$.

Now suppose $Sim = (C, M, R)$ is a simulator of the intended kind. From the menu function M we define an auxiliary function $B : \mathsf{S} \times \mathsf{Act} \to \mathbb{N}$ defined by

$$B(p, a) = |(a, i) \in M(p)|.$$

This function counts the number of a-transitions that can be chosen from the menu $M(p)$. Note that because M is a computable function it directly follows that B is also a computable function.

Now let $B(C(X), a) = k$ with $k \geq 1$. And define the process $Y \equiv \sum_{i=1}^{k+1} \mathrm{U2CM_n} \cdot b^i$ (see Remark 8.1). 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 $LTS/\leftrightarrows$.
- $n \notin K \Rightarrow B(C(Y), a) = B(C(X), a) = k$ by *completeness.*

Combining the last two implications, we have

$$B(C(Y), a) = k \Longleftrightarrow n \notin K,$$

which contradicts the fact that B is a computable function. So there can never exist a simulator that respects bismulation semantics. ■

To guide the intuition consider the processes $a \cdot P + a \cdot Q$ and $a \cdot P$ where P and Q are arbitrary ACP terms. Now a simulator respecting bisimulation should compute the same menu for these processes if P and Q are bisimilar because then also $a \cdot P + a \cdot Q$ and $a \cdot P$ are bisimilar (by compatibility). However this contradicts the fact that bisimulation equivalence is undecidable for (recursive) ACP terms [2, 15].

We conjecture that we can prove similar (negative) results as Theorem 8.1 in a setting of $LTS/\simeq$ (graph isomorphism), $LTS/\sim_{FT}$ (failure trace), $LTS/\sim_R$ (ready), $LTS/\sim_F$ (failure), $LTS/\sim_{CT}$ (completed trace). The definition of these equivalences can be found in [8]. However, there **do** exist simulators respecting $LTS/\sim_{RT}$ (ready trace) and $LTS/\sim_T$ (trace). In [10] the following theorem is proven.

THEOREM 8.2 *There exists a simulator Sim for language Terms*(ACP) respecting $LTS/\sim_{RT}$ *(ready trace) and there exists a simulator* respecting $LTS/\sim_T$ *(trace).*

The idea behind this result is that in trace semantics, we do not have to decide whether or not the menu $M(a \cdot P + a \cdot Q)$ consists of one or two choices. Via the well known trace law $a \cdot P + a \cdot Q = a \cdot (P + Q)$ we can always compute deterministic menus: $M(a \cdot P + a \cdot Q) = M(a \cdot (P + Q)) = \{(a, 1)\}$ without running into decidability problems. Similar arguments can be given for ready trace semantics. However, we could not find simulators respecting $LTS/\sim_{RT}$ or $LTS/\sim_T$ that run in polynomial time.

9. SUMMARY AND CONCLUSION

Table 2 summerizes the results of this paper: it shows the existence (Y) or *non* existence (N) of simulators respecting (being sound, complete and compatible with respect to) the well-known process models that have emerged from process theory [8]. 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 [8] and respectively 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).

From the first row in the table above one can make up there are simulators that are sound with respect to all the models mentioned in the table. This depends on Proposition 7.1 which states that the simulator given in Example 4.1 is sound with I (graph isomorphism). (This directly implies that this simulator is also sound with respect to all the other models in the table). Furthermore we know by Proposition 4.1 that this simulator runs in polynomial time which is denoted by P in the table.

In the second row one can see that we could not find efficient simulators that are sound and complete with respect to one of the models in the table.

In the third row it is shown that we can prove with a reasonable restriction, i.e. the compatibility criterion, that there do not exist simulators that are sound and complete with bisimulation semantics (denoted by B in the table).

All these observations together confirm our belief that an efficient simulator must be less identifying than bisimulation and maybe even less identifying than graph isomorphism.

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. Further I would like to thank Willem Jan Fokkink, Steven Klusener, Alban Ponse and Frits Vaandrager for feedback and suggestions. Finally, I would like to thank an anonymous referee for spotting bugs in, and for his helpful comments on, a previous draft of this paper.

REFERENCES

[1] S. Abramsky. Observation equivalence as a testing equivalence. *Theoretical Computer Science*, 53:225–241, 1987.

[2] 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.

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

[4] 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*

TABLE 2. Summary.

	I	B	RT	FT	R	F	CT	T
sound	Y (7.1) P (4.1)	Y (7.1) P (4.1)	Y (7.1) P (4.1)	Y (7.1) P (4.1)	Y (7.1) P (4.1)	Y (7.1) P (4.1)	Y (7.1) P (4.1)	Y (7.1) P (4.1)
sound & complete	Y? ¬P	Y (7.1) ¬P	Y (8.2) ¬P	Y? ¬P	Y? ¬P	Y? ¬P	Y? ¬P	Y (8.2) ¬P
sound & complete & compatible	N? ¬P	N (8.1)	Y (8.2) ¬P	N? ¬P	N? ¬P	N? ¬P	N? ¬P	Y (8.2) ¬P

Specification, Testing and Verification, Lecture Notes in Computer Science, pp. 287–302. North-Holland, Amsterdam, 1990.

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

[6] P. H. J. van Eijk. The design of a simulator tool. In van Eijk et al. [7], pp. 351–390.

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

[8] 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*, pp. 278–297. Springer-Verlag, Berlin, 1990.

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

[10] H. Korver. A theory of simulator tools. Report CS 9302, CWI, Amsterdam, 1993.

[11] R. Milner. *Communication and Concurrency.* Prentice-Hall International, Englewood Cliffs, NJ, 1989.

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

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

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

[15] 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, Berlin, 1989.

[16] J. Tretmans. HIPPO: a LOTOS simulator. In van Eijk *et al.* [7], pp. 391–396.

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

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