Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

J.F. Groote, F.W. Vaandrager

Structured operational semantics and bisimulation as a congruence

Computer Science/Department of Software Technology Report CS-R8845 November

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).

Structured Operational Semantics and Bisimulation as a Congruence

Jan Friso Groote
Frits Vaandrager
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

In recent years a large number of (concurrent) languages have been provided with an operational semantics using Plotkin's structural approach (SOS). In this paper the question is considered in which cases a transition system specification in Plotkin style has 'good' properties and deserves the predicate 'structured'. The discussion takes place in a setting of labelled transition systems. The states of the transition systems are terms generated by a single sorted signature and the transitions between states are defined by structural induction on abstract syntax. It is argued that in this setting it is natural to require that strong bisimulation equivalence is a congruence on the states of the transition systems. A general format, called the *tyft/tyxt* format, is presented for the inductive rules in a transition system specification, such that bisimulation is always a congruence when all the rules fit into this format. With a series of examples it is demonstrated that the *tyft/tyxt* format cannot be generalized in any obvious way. Another series of examples illustrates the usefulness of our congruence theorem. Briefly we touch upon the issue of modularity of transition system specifications. It is argued that certain pathological *tyft/tyxt* rules (the ones which are not *pure*) can be disqualified because they behave badly with respect to modularisation. Next we address the issue of full abstraction. We characterize the completed trace congruence induced by the operators in pure *tyft/tyxt* format as *2-nested simulation equivalence*. The pure *tyft/tyxt* format includes the format given by DE SIMONE [29, 30] but is incomparable to the GSOS format of BLOOM, ISTRAIL & MEYER [10]. However, it turns out that 2-nested simulation equivalence strictly refines the completed trace congruence induced by the GSOS format.

Key Words and Phrases: Structured Operational Semantics (SOS), transition system specifications, compositionality, labelled transition systems, bisimulation, congruence, process algebra, *tyft/tyxt* rules, modularity of transition system specifications, full abstraction, testing, nested simulations, Hennessy-Milner logic, GSOS rules.
1985 Mathematical Subject Classification: 68Q05, 68Q55.
1980 Mathematical Subject Classification: 68B10.
1982 CR Categories: D.3.1, F.1.1, F.3.2, F.4.3.
Note: The research of the authors was supported by ESPRIT project no. 432, An Integrated Formal Approach to Industrial Software Development (METEOR), and by RACE project no. 1046, Specification and Programming Environment for Communication Software (SPECS).

1. INTRODUCTION

In [26, 27] PLOTKIN advocates a simple method for giving operational semantics to programming languages. The method, which is often referred to as *SOS* (for *Structured Operational Semantics*), is based on the notion of transition systems. The states of the transition systems are elements of some formal language that, in general, will extend the language for which one wants to give an operational semantics. The main idea of the method is that the transitions between states are defined by structural induction on abstract syntax. Plotkin did not give a formal definition of his method, i.e. a definition of the type of expressions which in general are allowed as states, a format for the inductive rules and a formal definition which says when a transition system specification is 'structured'. Probably, he didn't see any reason for giving such a definition because it always would have been either too

Report CS-R8845
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

restrictive, excluding some important applications, or too general, declaring certain specifications to be structured even though they do not deserve this predicate at all.

In recent years a large number of (concurrent) languages have been provided with an operational semantics in terms of transition systems defined with structural induction on abstract syntax. We think that therefore it might be worthwhile to consider in more detail the questions how expressive different classes of transition system specifications are and in which cases transition system specifications have 'good' properties and can be called structured.

Before one can investigate the question what types of transition system specifications have 'good' properties and can be called structured, one has to say what are the 'good' properties one is interested in. Most people will agree that the semantics generated by a structured transition system specification should be compositional in some sense. We want to consider the question whether compositionality (under the interpretation that the transitions associated to a phrase are to be a function of the transitions associated to its immediate components) is a sufficient condition for calling a transition system specification structured. Does a compositional transition system specification have the nice properties one would like it to have? We think that in general this is not the case.

In order to make our point, we restrict our attention to a specific type of transition systems: transitions are labelled and as states we have ground terms generated by a single sorted signature. This is an important subcase: the operational semantics of languages like CCS [21], TCSP [24], ACP [13] and MEIJE [11] has been described in essentially this way. However, there are also examples of transition system specifications where the set of states is not specified by a single sorted signature, for instance the semantics for CSP as presented in PLOTKIN [27] and the semantics for POOL of AMERICA, DE BAKKER, KOK & RUTTEN [2]. We hope that the insights derived from our analysis of a basic case will somehow generalize to more general settings.

A fundamental equivalence on the states of a labelled transition system is the strong bisimulation equivalence of PARK [25]. Strong bisimulation equivalence seems to be the *finest* extensional behavioural equivalence one would want to impose, i.e. two states of a transition system which are bisimilar cannot be distinguished by external observation. This means that from an observational point of view, the transition systems generated by the SOS approach are too concrete as semantical objects. The objects that really interest us will be *abstract* transition systems where the states are bisimulation equivalence classes of terms, or maybe something even more abstract. If bisimulation is not a congruence then this means that the function that returns the transitions associated to a phrase when given the transitions associated to its immediate components, depends on properties of the transition system which are generally considered to be irrelevant, such as the specific names of states. Hence we think that a transition system specification which leads to transition systems for which bisimulation is not a congruence should not be called structured: possibly it is compositional on the level of (concrete) transition systems but it is not compositional on the more fundamental level of transition systems modulo bisimulation equivalence.

This brings us to the first main question of this paper which is to find a format, as general as possible, for the inductive rules in a transition system specification, such that bisimulation is always a congruence when all the rules have this format. We proceed in a number of steps.

In section 2 of the paper definitions are given of some basic notions like signature, term and substitution. Section 3 contains a formal definition of the notion of a transition system specification (TSS). In section 4 it is described how a TSS determines a transition system. Moreover the fundamental notion of strong bisimulation is introduced.

The real work starts in section 5, where we present a general format, called the *tyft/tyxt* format, for the inductive rules in a TSS and prove that bisimulation is always a congruence when all inductive rules have this format. With a series of examples it is demonstrated that this format cannot be generalized in any obvious way.

Section 6 contains some applications of our congruence theorem. We think that our result will be useful in many situations because it allows one to see immediately that bisimulation is a congruence. Thus it generalizes and makes less *ad hoc* the congruence proofs in [5, 23], and elsewhere. If the rules

in a TSS do not fit in our format then there is a good chance that something will be wrong: either bisimulation is not a congruence right away or the congruence property will get lost if more operators and rules are added.

A very natural and important operation on transition system specifications is to take their componentwise union. Given two specifications P_0 and P_1, let $P_0 \oplus P_1$ denote this union. A desirable property to have is that the outgoing transition of states in the transition system associated to P_0 are the same as the outgoing transitions of these states in the extended system $P_0 \oplus P_1$. This means that $P_0 \oplus P_1$ is a 'conservative extension' of P_0: any property which has been proved for the states in the old transition system remains valid (for the old states) in the enriched system. In section 7 we show that most of the *tyft/tyxt* rules (the rules which are *pure*) behave fine under modularisation. Rules that are not pure behave badly under modularisation, but fortunately these rules are quite pathological and we have never seen an application in which they are used. Hence, they can be discarded without great pain.

A central idea in the theory of concurrency is that processes which cannot be distinguished by observation, should be identified: the process semantics should be *fully abstract* with respect to some notion of testing (see [12]). Mostly one takes the position that the observations one can make on a process include its *completed traces*. A completed trace is a (finite) maximal sequence of actions which can be performed by a process. Two processes are *completed trace congruent* with respect to some format of rules if they yield the same completed traces in any context that can be built from operations defined in this format. The main result of section 8 of this paper is a characterization, valid for image finite transition systems, of the completed trace congruence induced by the pure *tyft/tyxt* format as *2-nested simulation equivalence*. On the domain of image finite transition systems, 2-nested simulation coincides with the equivalence induced by the Hennessy-Milner logic formulas [17] with no [] in the scope of a $\langle\rangle$. Consequently the following two trees, which are not bisimilar, cannot be distinguished by operators defined with pure *tyft/tyxt* rules:

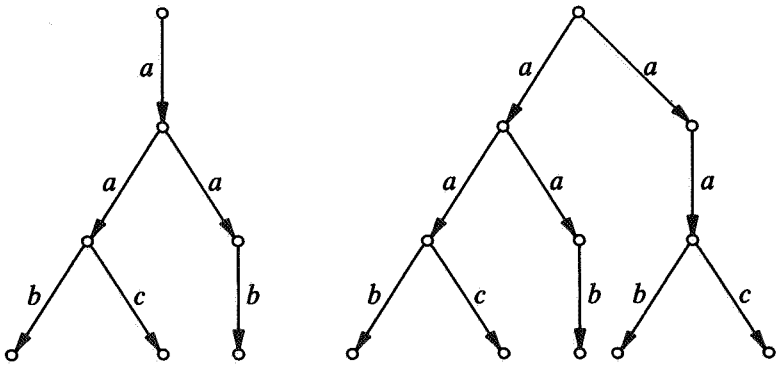


FIGURE 1. Pure *tyft/tyxt* congruent but not bisimilar

In section 9 we give an extensive comparison of our format, the format proposed by DE SIMONE [29, 30] and the GSOS format of BLOOM, ISTRAIL & MEYER [10]. Roughly speaking, the situation is as displayed in figure 2. The GSOS format and the pure *tyft/tyxt* format both generalize the format of De Simone. The GSOS format and our format are incomparable since the GSOS format allows negations in the premises, whereas all our rules are positive. On the other hand we allow for rules that give operators a lookahead and this is not allowed by the GSOS format. A simple example in [10] shows that the combination of negation and lookahead is inconsistent. The point where the two formats diverge is characterized by the rules which fit into the GSOS format but which contain no negation. We call the corresponding format *positive GSOS*.

BLOOM, ISTRAIL & MEYER [10] proved that the completed trace congruence induced by the GSOS format can be characterized by the class of Hennessy-Milner logic formulas in which only F may

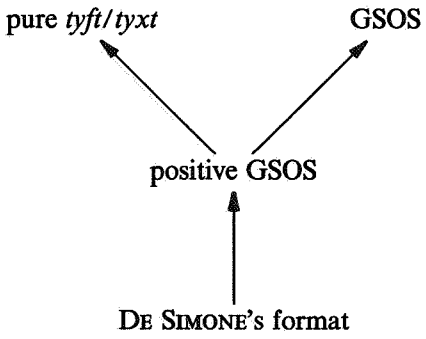


FIGURE 2

occur in the scope of a []. LARSEN & SKOU [20] in turn showed that the equivalence induced by this class of logical formulas can be characterized as *2/3-bisimulation*. From these results we can conclude quite directly that the *tyft/tyxt* format can make more distinctions between processes than the GSOS format: 2-nested simulation refines 2/3-bisimulation. Now, interestingly, it turns out that the completed trace congruence induced by the *positive* GSOS format is also 2/3-bisimulation equivalence. So although it may be the case that in the general GSOS format can be used to define certain operations which cannot be defined using positive rules only, the use of negations in the definition of operators does not introduce any new distinctions between processes!

The notion of testing associated with the (positive) GSOS format allows one to observe *traces* of processes, to detect *refusals* and to make *copies* of processes at every moment. Our format allows one in addition to test whether some action is possible in the future: operators can have a *lookahead*. This can be seen as a weak form of *global testing* (cf. [1]).

A notable difference between the GSOS format and our format is that the GSOS format always leads to a computably finitely branching transition relation whereas our format does not. We argue that the statement of BLOOM, ISTRAIL & MEYER [10] that any 'reasonably structured' specification should induce a computably finitely branching transition relation, is too strong and discards a large number of interesting applications.

ACKNOWLEDGEMENTS. We want to thank Bard Bloom for a very interesting and stimulating correspondence. Discussions with him had a pervasive influence on the contents of this paper. We also thank Rob van Glabbeek for many useful comments and inspiring discussions.

2. SIGNATURES, TERMS AND SUBSTITUTIONS

In this paper we will work with a very simple notion of a signature. Only one sort is allowed and there is no overloading. Throughout this paper we assume the presence of a countably infinite set V of *variables* with typical elements $x,y,z...$

2.1. DEFINITION. A *(single sorted) signature* is a structure $\Sigma=(F,r)$ where:
- F is a set of *function names* disjoint with V,
- $r:F\to\mathbb{N}$ is a *rank function* which gives the arity of a function name; if $f\in F$ and $r(f)=0$ then f is called a *constant name*.

2.2. DEFINITION. Let $\Sigma=(F,r)$ be a signature. Let $W \subseteq V$ be a set of variables. The set of Σ-*terms* over W, notation $T(\Sigma,W)$, is the least set satisfying:
- $W \subseteq T(\Sigma,W)$,
- if $f \in F$ and $t_1,..,t_{r(f)} \in T(\Sigma,W)$, then $f(t_1,..,t_{r(f)}) \in T(\Sigma,W)$.

$T(\Sigma,\varnothing)$ is abbreviated by $T(\Sigma)$; elements from $T(\Sigma) \subseteq T(\Sigma,V)$ are called *closed* or *ground terms*. $Var(t) \subseteq V$ is the set of variables in a term $t \in T(\Sigma,V)$.

2.3. DEFINITION. Let $\Sigma=(F,r)$ be a signature. A *substitution* σ is a mapping in $V \to T(\Sigma,V)$. A substitution σ is extended to a mapping $\sigma: T(\Sigma,V) \to T(\Sigma,V)$ in a standard way by the following definition:
- $\sigma(f(t_1,..,t_{r(f)})) = f(\sigma(t_1),..,\sigma(t_{r(f)}))$ for $f \in F$ and $t_1,..,t_{r(f)} \in T(\Sigma,V)$.

If σ and ρ are substitutions, then the substitution $\sigma \circ \rho$ is defined by:

$$\sigma \circ \rho(x) = \sigma(\rho(x)) \quad \text{for } x \in V.$$

2.4. NOTE. Observe that we have the following identities:

$$\sigma \circ \rho(t) = \sigma(\rho(t)) \qquad t \in T(\Sigma,V)$$
$$\sigma(t) = t \qquad \text{for } t \in T(\Sigma)$$

3. TRANSITION SYSTEM SPECIFICATIONS

In this section a formal definition is given of the notion of a transition system specification. Also the notion of a proof of a transition from such a specification is defined.

3.1. DEFINITION. A *transition system specification (TSS)* is a 3-tuple (Σ,A,R) with Σ a signature, A a set of *labels* and R a set of *rules* of the form:

$$\frac{\{t_i \xrightarrow{a_i} t_i' \mid i \in I\}}{t \xrightarrow{a} t'}$$

where I is a finite index set, $t_i,t_i',t,t' \in T(\Sigma,V)$ and $a_i,a \in A$ for $i \in I$. If r is a rule satisfying the above format, then the elements of $\{t_i \xrightarrow{a_i} t_i' \mid i \in I\}$ are called the *premises* of r and $t \xrightarrow{a} t'$ is called the *conclusion* of r. A rule of the form $\frac{\varnothing}{t \xrightarrow{a} t'}$ is called an *axiom*, which, if no confusion can arise, is also written as $t \xrightarrow{a} t'$. An expression of the form $t \xrightarrow{a} t'$ with $a \in A$ and $t,t' \in T(\Sigma,V)$ is called a *transition* (labelled with a). The letters $\phi,\psi,\chi,..$ will be used to range over transitions. The notions 'substitution', '*Var*' and 'closed' extend to transitions and rules as expected.

3.2. DEFINITION. Let $P=(\Sigma,A,R)$ be a TSS. A *proof* of a transition ψ from P is a finite, upwardly branching tree of which the nodes are labelled by transitions $t \xrightarrow{a} t'$ with $t,t' \in T(\Sigma,V)$ and $a \in A$, such that:
- the root is labelled with ψ,
- if χ is the label of a node q and $\{\chi_i \mid i \in I\}$ is the set of labels of the nodes directly above q, then there is a rule $\frac{\{\phi_i \mid i \in I\}}{\phi}$ in R and a substitution $\sigma: V \to T(\Sigma,V)$ such that $\chi=\sigma(\phi)$ and $\chi_i=\sigma(\phi_i)$ for $i \in I$.

If a proof of ψ from P exists, we say that ψ is *provable* from P, notation $P \vdash \psi$. A proof is *closed* if it only contains closed transitions.

3.3. LEMMA. *Let $P=(\Sigma,A,R)$ be a TSS, let $a\in A$ and let $t,t'\in T(\Sigma)$ such that $P\vdash t\xrightarrow{a}t'$. Then $t\xrightarrow{a}t'$ is provable by a closed proof.*

PROOF. As $P\vdash t\xrightarrow{a}t'$ there is a proof tree T for $t\xrightarrow{a}t'$. Define the substitution $\sigma:V\to T(\Sigma)$ by $\sigma(x)=t$ for all $x\in V$. Applying σ to all transitions in the proof T of $t\xrightarrow{a}t'$ yields a tree T' containing only closed transitions. Now one can easily check that T' is a proof of $t\xrightarrow{a}t'$. □

TSS's have been used mainly as a tool to give operational semantics to (concurrent) programming languages. As a running example we therefore present below a TSS for a simple process language.

3.4. EXAMPLE. Let $Act=\{a,b,c,..\}$ be a given set of *actions*. We consider the signature $\Sigma(\mathrm{BPA}_\delta)$ (Basic Process Algebra with δ and ϵ) as introduced in [31]. $\Sigma(\mathrm{BPA}_\delta)$ contains constants a for each $a\in Act$, a constant δ that stands for *deadlock*, comparable to NIL in CCS and STOP in TCSP, and a constant ϵ that denotes the *empty process*, a process that terminates immediately and successfully. It is comparable to SKIP in TCSP and CCS. Furthermore the signature contains binary operators $+$ (*alternative composition*) and $\cdot$ (*sequential composition*). As labels of transitions we take elements of $Act_\surd=Act\cup\{\surd\}$. Here $\surd$ (pronounce 'tick') is a special symbol used to denote the action of successful termination. At the end of a process this action indicates to the environment that execution has finished.
Define the TSS $P(\mathrm{BPA}_\delta)$ as $(\Sigma(\mathrm{BPA}_\delta),Act_\surd,R(\mathrm{BPA}_\delta))$ where $R(\mathrm{BPA}_\delta)$ is defined below in table 1. In the table a ranges over $Act_\surd$, unless further restrictions are made. Infix notation is used for the binary function names.

1.	$a\xrightarrow{a}\epsilon$	$a\neq\surd$	2.	$\epsilon\xrightarrow{\surd}\delta$
3.	$\dfrac{x\xrightarrow{a}x'}{x+y\xrightarrow{a}x'}$		4.	$\dfrac{y\xrightarrow{a}y'}{x+y\xrightarrow{a}y'}$
5.	$\dfrac{x\xrightarrow{a}x'}{x\cdot y\xrightarrow{a}x'\cdot y}$	$a\neq\surd$	6.	$\dfrac{x\xrightarrow{\surd}x'\quad y\xrightarrow{a}y'}{x\cdot y\xrightarrow{a}y'}$

TABLE 1

One can easily check that the tree in figure 3 constitutes a proof of the transition $(\epsilon\cdot(a+b))\cdot c\xrightarrow{a}\epsilon\cdot c$ from $P(\mathrm{BPA}_\delta)$.

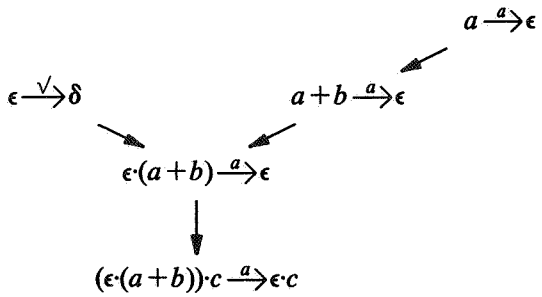


FIGURE 3

3.4.1. REMARK. Even though (extensions of) the signature of BPA§ occur at a number of places, the rules of table 1 seem to be new. VRANCKEN [31] does not use inductive rules to give semantics to BPA§. Instead he defines the operations of BPA§ directly on *process graphs*. In BAETEN & VAN GLABBEEK [5] there are no transitions labelled with $\surd$. Instead they use a unary termination predicate $\downarrow$. The analogue of our rule 6 in their setting is:

$$\frac{x\downarrow,\ y \xrightarrow{a} y'}{x \cdot y \xrightarrow{a} y'}$$

Such a rule does not fit in our framework because we do not allow for this type of predicates in a transition system specification. We think however that our rules are simpler and have a number of advantages. For instance, Baeten & Van Glabbeek have to add an additional clause to the definition of bisimulation equivalence in order to deal with termination. In our approach this is not needed: when describing the intended semantics we can just work with strong bisimulation equivalence.

3.5. EXAMPLE. Our next example shows that the range of applications of TSS's is not restricted to the area of operational semantics: every Term Rewriting System (TRS) can be viewed as a TSS. Unfortunately, the class of TSS's which correspond to TRS's and the class of TSS's for which bisimulation is a congruence have an intersection which is almost empty. A *Term Rewriting System (TRS)* is defined as a pair (Σ_0,R_0) with Σ_0 a signature and R_0 a set of *reduction-* or *rewrite rules* of the form $r:(t,s)$ with r the *name* of the rewrite rule and $t,s \in T(\Sigma_0,V)$. Here, t contains at least one function name and $Var(s) \subseteq Var(t)$.
A TRS can be viewed as a TSS (Σ,A,R). Take $\Sigma=\Sigma_0$ as the signature and define the alphabet A as the set of all names r of rules $r:(t,s)\in R_0$. R contains for every $r:(t,s)\in R_0$ a rule:

$$t \xrightarrow{r} s$$

and for every function name f in Σ rules:

$$\frac{x \xrightarrow{r} y}{f(x_1,..,x,..,x_{r(f)}) \xrightarrow{r} f(x_1,..,y,..,x_{r(f)})}$$

to allow reductions in contexts. One can easily prove that there is a *one step rewrite* $t \rightarrow_r s$ in the TRS (see [19] for a definition) iff the corresponding TSS proves $t \xrightarrow{r} s$.

4. TRANSITION SYSTEMS AND STRONG BISIMULATION EQUIVALENCE

An operational semantics makes use of some sort of (abstract) machine and describes how these machines behave. Often one takes as machines simply nondeterministic automata in the sense of classical automata theory, also called labelled transition systems [18].

4.1. DEFINITION. A (*nondeterministic*) *automaton* or *labelled transition system (LTS)* is a structure $(S,A,\rightarrow)$ where:
- S is a set of *states*,
- A is an *alphabet*,
- $\rightarrow \subseteq S \times A \times S$ is a *transition relation.*

Elements $(s,a,s') \in \rightarrow$ are called *transitions* and will be written as $s \xrightarrow{a} s'$. The intended interpretation is that from state s the machine can do an action a and thereby get into state s'.

4.1.1. REMARK. Often transition systems are provided with an additional fourth component: the *initial state*. For our purpose it has some small technical advantages to work with transition systems that do not contain this ingredient. All considerations of this paper can trivially be extended to transition systems with initial state.

The notion of strong bisimulation equivalence as defined below is from PARK [25].

4.2. DEFINITION. Let $\mathcal{C}=(S,A,\rightarrow)$ be a labelled transition system. A relation $R \subseteq S \times S$ is a (*strong*) *bisimulation* if it satisfies:
1. whenever $s\,R\,t$ and $s \xrightarrow{a} s'$ then, for some $t' \in S$, also $t \xrightarrow{a} t'$ and $s'R\,t'$,
2. conversely, whenever $s\,R\,t$ and $t \xrightarrow{a} t'$ then, for some $s' \in S$, also $s \xrightarrow{a} s'$ and $s'R\,t'$.

Two states $s,t \in S$ are *bisimilar* in $\mathcal{C}$, notation $\mathcal{C}{:}s \leftrightarrows t$, if there exists a bisimulation containing the pair (s,t). Note that bisimilarity is indeed an equivalence relation on states.

4.3. DEFINITION (*TSS's, transition systems and bisimulation*). Let $P=(\Sigma,A,R)$ be a TSS. The transition system $TS(P)$ specified by P is given by:

$$TS(P) = (T(\Sigma),A,\rightarrow_P).$$

Here the relation $\rightarrow_P \subseteq T(\Sigma)\times A \times T(\Sigma)$ is defined by:

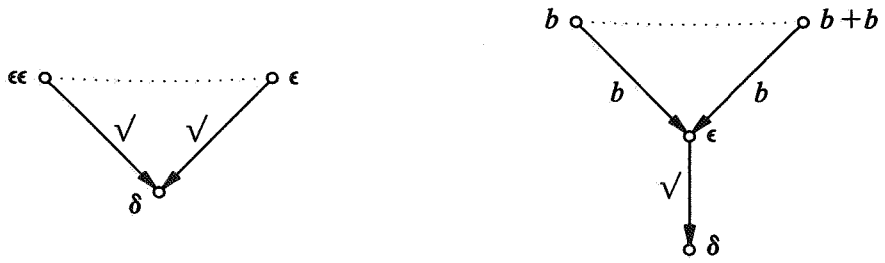
$$t \xrightarrow{a}_P t' \quad \Leftrightarrow \quad P \vdash t \xrightarrow{a} t'.$$

We say that two terms $t,t' \in T(\Sigma)$ are (*P-)bisimilar*, notation $t \leftrightarrows_P t'$, if $TS(P){:}t \leftrightarrows t'$. We write $t \leftrightarrows t'$ if it is clear from the context what P is. Note that $\leftrightarrows_P$ is also an equivalence relation.

4.4. EXAMPLE. For the TSS P(BPA$_\delta$) of example 3.4 we can derive the identities (a)-(e) below. In (f) it is shown that the left distributivity of $\cdot$ over $+$ does not hold in bisimulation semantics. Like in regular algebra we will often omit the $\cdot$ in a product $x{\cdot}y$ and we take $\cdot$ to be more binding than $+$. Missing brackets in expressions xyz and $x+y+z$ associate to the right.

(a)	$\epsilon\epsilon \leftrightarrows \epsilon$	(d)	$b\epsilon \leftrightarrows b$
(b)	$b \leftrightarrows b+b$	(e)	$\epsilon b \leftrightarrows b$
(c)	$(\epsilon a+\epsilon b)(cd\delta+\delta) \leftrightarrows (a(c+\delta)d+bc(d+d))\delta$	(f)	$abc+abd \not\leftrightarrows a(bc+bd)$

The parts of the automaton belonging to (a),(b),(c) and (f) are drawn in figure 4-6. A dotted line indicates that a pair of states is in the bisimulation relation. Furthermore, a state is always related to itself. In showing that two states are related, only the states that can be reached from these states are relevant and therefore only these states are drawn.



(a) (b)

FIGURE 4

In figures 5/6 two separate automata are drawn instead of a combined one, to make the pictures clearer.
In figure 6 the states $a(bc+bd)$ and $\epsilon(bc+bd)$ in the right transition system cannot be related to any of the states in the left transition system.

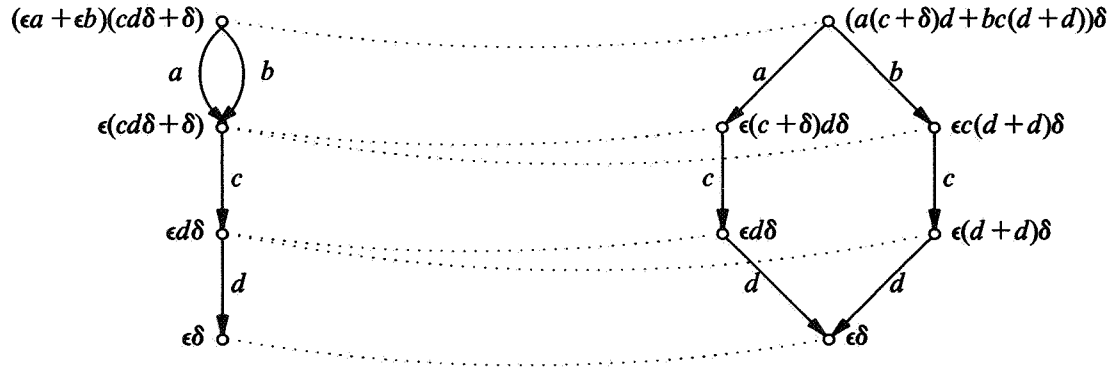


FIGURE 5 (example 4.4(c))

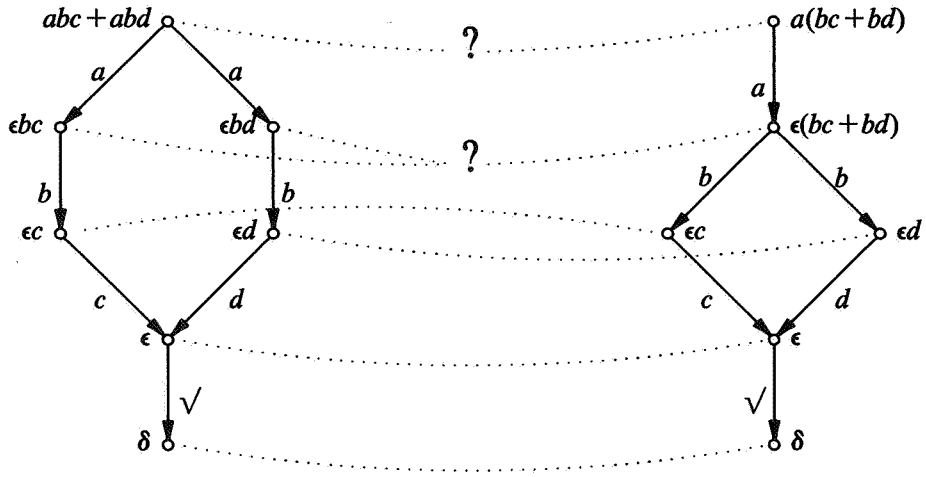


FIGURE 6 (example 4.4(f))

5. COMPOSITIONAL TRANSITION SYSTEM SPECIFICATIONS

TSS's do not always generate automata for which strong bisimulation is a congruence. A number of examples will follow in the sequel. But if the rules in TSS satisfy the format below (and an additional small technical requirement is met), strong bisimulation will turn out to be a congruence.

5.1. DEFINITION. Let $\Sigma=(F,r)$ be a signature and let $P=(\Sigma,A,R)$ be a TSS. A rule in R is in *tyft format* if it has the following form:

$$\frac{\{t_i \xrightarrow{a} y_i \mid i\in I\}}{f(x_1,..,x_{r(f)}) \xrightarrow{a} t}$$

with I a finite index set, f a function name from F, x_i $(1\leqslant i\leqslant r(f))$ and y_i $(i\in I)$ are all different variables from V, $a_i,a\in A$ and $t_i,t\in T(\Sigma,V)$ for $i\in I$.

A rule in R is in *tyxt format* if it has the following form:

$$\frac{\{t_i \xrightarrow{a} y_i \mid i\in I\}}{x \xrightarrow{a} t}$$

with I a finite index set, x,y_i $(i\in I)$ all different variables from V, $a_i,a\in A$ and $t_i,t\in T(\Sigma,V)$ for $i\in I$. P is in *tyft/tyxt format* if all the rules in R are in *tyft/tyxt* format. A transition system is called *tyft/tyxt specifiable* if it can be specified by a TSS in *tyft/tyxt* format.

5.2. NOTE. Observe that there does not have to be any relation at all between the premises and the conclusions in a rule satisfying our format. In fact our format explicitly requires the absence of certain relations between occurrences of variables in the premises and in the conclusion. Note that not only the TSS $P(\mathrm{BPA}_{\delta})$ of example 3.4 is in *tyft/tyxt* format, but also any TSS obtained from $P(\mathrm{BPA}_{\delta})$ by dropping some arbitrary rules. The transition system specifications related to term rewriting systems (see example 3.5) are in general not in *tyft/tyxt* format.

5.3. EXAMPLE. Below we describe a TSS that models a simple typewriter that can be used to type strings and that has the option to delete the last character of the typed string using 'backspace'. The signature consists of the binary function name $\star$ denoting concatenation, constant names λ (empty string) and $a,b,..,y,z$. As alphabet we take $A=\{a,b,..,y,z,\Delta\}$. Here, Δ stands for a backspace. Rules for the typewriter can be given as follows:

$$x \xrightarrow{a} x\star a \qquad \text{for } a\in\{a,b,..,y,z\}$$

$$a \xrightarrow{\Delta} \lambda \qquad \text{for } a\in\{a,b,..,y,z\}$$

$$x\star a \xrightarrow{\Delta} x \qquad \text{for } a\in\{a,b,..,y,z\}$$

This description of the typewriter is not in *tyft/tyxt* format, because the lhs of the last axiom contains two function names. A TSS for the typewriter in *tyft/tyxt* format is more involved. We need an auxiliary label *empty*, which denotes that an expression consists of the empty string. We also need more rules:

$$x \xrightarrow{a} x\star a \qquad \text{for } a\in\{a,b,..,y,z\}$$

$$a \xrightarrow{\Delta} \lambda \qquad \text{for } a\in\{a,b,..,y,z\}$$

$$\lambda \xrightarrow{empty} \lambda$$

$$\frac{x \xrightarrow{\Delta} x'}{y\star x \xrightarrow{\Delta} y\star x'}$$

$$\frac{x \xrightarrow{a} x' \quad y \xrightarrow{empty} y'}{x\star y \xrightarrow{a} x'} \qquad \text{for } a\in\{empty,\Delta\}$$

We leave it as an exercise to the reader to show that the identities $\lambda\star t \leftrightarrows t\star\lambda \leftrightarrows t$ and $(t_1\star t_2)\star t_3 \leftrightarrows t_1\star(t_2\star t_3)$ with t,t_1,t_2,t_3 closed terms over the signature of the typewriter, hold for this TSS.

5.4. Circularity. A TSS with the rule:

$$\frac{f(x,y_2)\xrightarrow{a} y_1 \quad g(x',y_1)\xrightarrow{b} y_2}{x \xrightarrow{c} x'}$$

can be in *tyft/tyxt* format. However, we have a sort of circular reference. The particular form of y_1 will, in general, depend on $f(x,y_2)$ and thus on y_2 while y_2 depends on $g(x',y_1)$ and thus on y_1. We will exclude this type of dependencies, as they give rise to complicated TSS's. For this purpose the notion of a *dependency graph* is introduced.

5.4.1. DEFINITION. Let $P=(\Sigma,A,R)$ be a TSS. Let $S=\{t_i \xrightarrow{a_i} t_i' \mid i\in I\}$ be a set of transitions of P. The *dependency graph* of S is a directed (unlabelled) graph with:
- Nodes: $\bigcup_{i\in I} Var(t_i \xrightarrow{a_i} t_i')$,
- Edges: $\{<x,y> \mid x\in Var(t_i),\ y\in Var(t_i')$ for some $i\in I\}$.

A set of transitions is called *circular* if its dependency graph contains a cycle. A rule is called *circular* if the set of its premises is circular. A set of rules is called *circular* if it contains a circular rule. Finally, a TSS is called *circular* if its set of rules is circular.

5.4.2. EXAMPLE. The dependency graph of the set of premises of the rule in section 5.4 is given in figure 7. The rule is circular since the graph clearly contains a cycle.

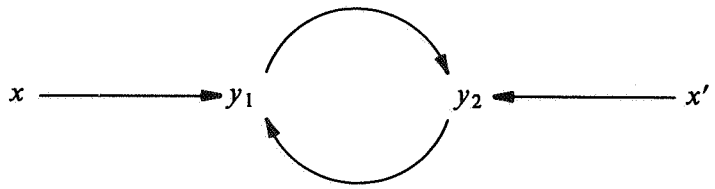


FIGURE 7

5.5. DEFINITION. Two TSS's P and P' are *equivalent* if $TS(P) = TS(P')$.

Hence, two TSS's are equivalent if they have the same signature, the same set of labels and if the sets of rules determine the same transition relation. The particular form of the rules is not important. In example 3.4 for instance, we can replace rule 6 of table 1 by the rule:

$$\frac{x \xrightarrow{\surd} \delta \quad y \xrightarrow{a} y'}{x\cdot y \xrightarrow{a} y'}$$

The resulting TSS $P'(\mathrm{BPA}_{\delta}^{\S})$ is equivalent to $P(\mathrm{BPA}_{\delta}^{\S})$. The reason for this is that whenever $P(\mathrm{BPA}_{\delta}^{\S})$ proves a transition of the form $t \xrightarrow{\surd} t'$, t' will be syntactically equal to δ. Observe that $P'(\mathrm{BPA}_{\delta}^{\S})$ is not in *tyft/tyxt* format. We will come back to this in section 5.13.

When dealing with closed terms, only the *tyft* format is necessary and the *tyxt* format is not needed. This is what the following lemma says.

5.6. LEMMA. *Let $P=(\Sigma,A,R)$ be a (non circular) TSS in tyft/tyxt format. Then there is an equivalent (non circular) TSS $P'=(\Sigma,A,R')$ in tyft format.*

PROOF. Let $\Sigma=(F,r_0)$. Define R' by:
- every *tyft* rule of R is in R',
- for every *tyxt* rule $r\in R$ and for every function name $f\in F$, r_f is in R', where r_f is obtained by substituting $f(x_1,..,x_{r_0(f)})$ for x in r with $\{x_1,..,x_{r_0(f)}\}\subseteq V-Var(r)$.

If the old *tyxt* rules were non circular, then the new rules will be non circular too and in *tyft* format. Suppose that $t \xrightarrow{a} t'$ is transition in $TS(P)$. Then, by definition of $TS(P)$ and lemma 3.3, there is a closed proof from P of this transition. Now one can easily see that this is also a proof for $t \xrightarrow{a} t'$ from P'. A similar argument gives that every transition of $TS(P')$ is also a transition of $TS(P)$. □

5.7. DEFINITION. Let $P=(\Sigma,A,R)$ be a TSS and let r be a rule in R. A variable in $Var(r)$ is called *free* if it does not occur in the left hand side of the conclusion or in the right hand side of a premise.

5.8. DEFINITION. Let $P=(\Sigma,A,R)$ be a TSS. A rule $r\in R$ is called *pure* if it is non circular and contains no free variables. The TSS P is *pure* if all its rules are pure.

5.9. LEMMA. *Let $P=(\Sigma,A,R)$ be a non circular TSS in tyft/tyxt format. Then there is an equivalent pure TSS $P'=(\Sigma,A,R')$ in tyft format.*

PROOF. By the previous lemma we can assume that P is in *tyft* format. Replace every rule with free variables by a set of new rules. The new rules are obtained by applying every possible substitution of closed terms for the free variables in the old rule. If the old rules were non circular and in *tyft* format then the new rules will be pure and in *tyft* format. Now, every closed proof T for a transition $t_1 \xrightarrow{a} t_2$ from P is also a proof for $t_1 \xrightarrow{a} t_2$ from P' and vice versa. □

We now come to the first main theorem of this paper. It says that strong bisimulation is a congruence for all operators defined using a non circular TSS in *tyft/tyxt* format.

5.10. THEOREM. *Let $\Sigma=(F,r)$ be a signature and let $P=(\Sigma,A,R)$ be a TSS. If P is non circular and in tyft/tyxt format then $\leftrightarrows_P$ is a congruence for all function names in F, i.e. for all function names f in F and all closed terms $u_i,v_i\in T(\Sigma)$ $(1\leqslant i\leqslant r(f))$:*

$$\forall i\ u_i \leftrightarrows_P v_i \quad\Rightarrow\quad f(u_1,..,u_{r(f)}) \leftrightarrows_P f(v_1,..,v_{r(f)}).$$

5.11. COUNTEREXAMPLES. Before we commence with the proof of this theorem, we present a number of examples which show that the condition in the theorem that the TSS is in *tyft/tyxt* format cannot be weakened in any obvious way. At present, we have no example to show that the condition that the TSS is non circular cannot be missed: we just have not been able to prove the theorem without it. However, circular TSS's are so exotic that we doubt whether they will ever be used. We can at least not think of an application. In section 7 it will be shown that circular rules are ill-behaved with respect to modularisation.

5.11.1. EXAMPLE. The first example shows that in general the variables in the left side of the arrow in the conclusion must all be different. The crucial part of the example is a rule that one could call a *syntactical tester*. In case of the alternative composition, it tests whether the left and right argument of the $+$ are syntactically identical. The TSS which we have in mind, is obtained by adding to $P(\mathrm{BPA}_\delta)$ the axiom: $x+x \xrightarrow{ok} \delta$. We then have $a \leftrightarrows a\epsilon$, but $a+a \not\leftrightarrows a+a\epsilon$ as a and $a\epsilon$ are not syntactically equal.

5.11.2. EXAMPLE. In general there may not appear more than one function name at the left of the transition predicate in the conclusion. Take the TSS $P(\mathrm{BPA}_\delta)$ extended with the axiom $ab \xrightarrow{ok} \delta$. As in example 4.4(b) $b \leftrightarrows b+b$, but in the new situation we do not have any more that $ab \leftrightarrows a(b+b)$ as $a(b+b)$ cannot do an initial ok-transition. Another example illustrating this point is obtained by adding the axiom $x+(y+z) \xrightarrow{ok} \delta$ to $P(\mathrm{BPA}_\delta)$. Again we have $b \leftrightarrows b+b$, but now it is not the case that $b+(b+b) \leftrightarrows b+b$. As a last example of this kind we mention the typewriter of section 5.3. The first specification is not in *tyft/tyxt* format, because it contains the axiom $x*a \xrightarrow{\Delta} x$ with $*$ and a function names. Now $\lambda*a \leftrightarrows a$ but $a*(\lambda*a) \not\leftrightarrows a*a$. This problem does not arise in the *tyft/tyxt* version of the typewriter.

5.11.3. EXAMPLE. Our next example shows that in the premises the right hand side of a transition may contain no function names. We can add *prefixing* operators $a{:}(\cdot)$ to $P(\mathrm{BPA}_\delta)$ for each $a \in Act$ and define the operational meaning of these operators with rules:

$$a{:}x \xrightarrow{a} x.$$

If we now add moreover the rule:

$$\frac{x \xrightarrow{a} \epsilon}{x \xrightarrow{ok} \delta}$$

we have problems because $a{:}\epsilon \not\leftrightarrows a{:}(\epsilon{\cdot}\epsilon)$ even though $\epsilon \leftrightarrows \epsilon{\cdot}\epsilon$ (see example 4.4(a)).

5.11.4. EXAMPLE. The variables at the right hand side of the arrows in the premises may in general not coincide. This is shown by adding the rule:

$$\frac{x \xrightarrow{a} y \quad x' \xrightarrow{a} y}{x{\cdot}x' \xrightarrow{ok} \delta} \quad a \neq \surd$$

to $P(\mathrm{BPA}_\delta)$. Now $a \leftrightarrows a\epsilon$, but $aa \not\leftrightarrows (a\epsilon)a$.

5.11.5. EXAMPLE. If variables in the left hand side of the conclusion and the right hand side of the premises coincide, problems can arise too. Add the rule:

$$\frac{x \xrightarrow{a} y}{x+y \xrightarrow{ok} \delta}$$

to $P(\mathrm{BPA}_\delta)$ and observe that $\epsilon\epsilon \leftrightarrows \epsilon$, but $a+\epsilon\epsilon \not\leftrightarrows a+\epsilon$.

5.12. We now will prove theorem 5.10.

PROOF. Let $\Sigma=(F,r)$ be a signature and let $P=(\Sigma,A,R_0)$ be a non circular TSS in *tyft/tyxt* format. We have to prove that $\leftrightarrows_P$ is a congruence. Let $R \subseteq T(\Sigma)\times T(\Sigma)$ be the least relation satisfying:

- $\leftrightarrows_P \subseteq R$,
- for all function names f in F and terms u_i,v_i in $T(\Sigma)$ (for $1 \leqslant i \leqslant r(f)$):

$$(\forall i\ u_i\, R\, v_i) \ \Rightarrow\ f(u_1,..,u_{r(f)})\, R\, f(v_1,..,v_{r(f)}).$$

It is enough to show that $R \subseteq \leftrightarrows_P$ because from that it immediately follows that $\leftrightarrows_P$ is a congruence for all f in F. In order to prove that $R \subseteq \leftrightarrows_P$ it is enough to show that R is a bisimulation. For reasons of symmetry it is even enough to show only one half of the transfer property: if $u\,R\,v$ and $u \xrightarrow{a}_P u'$ then there is a v' such that $v \xrightarrow{a}_P v'$ and $u'R\,v'$. If $u\,R\,v$ then by definition of R either $u \leftrightarrows_p v$ or, for some function name f in F: $u \equiv f(u_1,..,u_{r(f)})$ and $v \equiv f(v_1,..,v_{r(f)})$ with $u_i\,R\,v_i$ for all i. As $\leftrightarrows_p$ trivially satisfies the transfer property, only the second option needs to be checked. Summarizing, we have to prove the following statement:

Whenever $P \vdash f(u_1,..,u_{r(f)}) \xrightarrow{a} u'$ *and* $u_i\,R\,v_i$ *for* $1 \leqslant i \leqslant r(f)$ *then there is a* v' *such that* $P \vdash f(v_1,..,v_{r(f)}) \xrightarrow{a} v'$ *and* $u'R\,v'$.

Lemma 3.3 says that there is a proof for $f(u_1,..,u_{r(f)}) \xrightarrow{a} u'$ that only contains closed transitions. We will prove the statement with induction on the structure of this proof. Lemma 5.9 allows us to assume throughout the proof that the rules in R_0 are pure and in *tyft* format.

Basis. Transition $f(u_1,..,u_{r(f)}) \xrightarrow{a} u'$ has a proof tree consisting of a single node. Hence, there is an axiom r in R_0 and a substitution $\sigma{:}V \to T(\Sigma)$ such that $\sigma(r) = f(u_1,..,u_{r(f)}) \xrightarrow{a} u'$. This means that r is of the form $f(x_1,..,x_{r(f)}) \xrightarrow{a} t$ with $x_i \in V$ for $1 \leqslant i \leqslant r(f)$ and $t \in T(\Sigma,V)$ such that $\sigma(x_i)=u_i$ and $\sigma(t)=u'$. Now define a substitution $\sigma'{:}V \to T(\Sigma)$ by:

$$\sigma'(x) = \begin{cases} v_i & \text{if } x = x_i \text{ for } 1 \leqslant i \leqslant r(f) \\ \sigma(x) & \text{otherwise} \end{cases}$$

Note that this definition is correct as all x_i are different. Take $v' = \sigma'(t)$. The tree with a single node labelled $f(v_1,..,v_{r(f)}) \xrightarrow{a} v'$ is a proof as $\sigma'(r) = f(v_1,..,v_{r(f)}) \xrightarrow{a} v'$. We claim that $u' R v'$. By assumption $Var(t) \subseteq \{x_1,..,x_{r(f)}\}$ and $\sigma(x_i) R \sigma'(x_i)$ for $1 \leqslant i \leqslant r(f)$. Now the claim follows directly from the following fact.

FACT. *Let $t \in T(\Sigma, V)$ and let $\sigma, \sigma' : V \to T(\Sigma)$ be substitutions such that for all x in $Var(t)$: $\sigma(x) R \sigma'(x)$. Then $\sigma(t) R \sigma'(t)$.*

PROOF. Straightforward induction on the structure of t using the definition of R. □

Induction. Assume that $P \vdash f(u_1,..,u_{r(f)}) \xrightarrow{a} u'$ with a proof of depth $n > 1$. Let r be the last rule used in the proof. Assume that r is equal to:

$$\frac{\{t_i \xrightarrow{a_i} y_i \mid i \in I\}}{f'(x_1,..,x_{r(f')}) \xrightarrow{a} t}$$

It follows that: 1) $f' \equiv f$
2) $\sigma(x_i) = u_i$ for $1 \leqslant i \leqslant r(f)$
3) $\sigma(t) = u'$

Our aim is to use the rule r again in the proof of $f(v_1,..,v_{r(f)}) \xrightarrow{a} v'$ for some v' by finding a proper substitution σ'. Define:

$$\sigma'(x_i) = v_i \qquad \text{for } 1 \leqslant i \leqslant r(f)$$
$$\sigma'(x_i) = \sigma(x) \qquad \text{for } x \notin X \cup Y$$

Here $X = \{x_i \mid 1 \leqslant i \leqslant r(f)\}$ and $Y = \{y_i \mid i \in I\}$. Stepwise we will extend the definition of σ' to elements of Y in such a way that for all variables x in V: $\sigma'(x)$ is defined $\Rightarrow \sigma(x) R \sigma'(x)$. Consider the dependency graph G of the premises of r. Call a node of G *coloured* if σ' is defined for this node (cq. variable). So initially we are in a state where all nodes are coloured except for the ones in Y. We will colour these nodes one by one. In the process the invariant will be preserved that whenever a node is coloured, all its predecessors are coloured too.

A term or transition is called coloured if all the variables contained in it are coloured. Hence, initially none of the transitions in $\{t_i \xrightarrow{a_i} y_i \mid i \in I\}$ is coloured. In the process of colouring variables we also preserve the invariant that whenever a transition $\psi \in \{t_i \xrightarrow{a_i} y_i \mid i \in I\}$ is coloured, i.e. σ' is defined for all variables of ψ, there exists a proof from P of $\sigma'(\psi)$.

Now we first observe that with a complete colouring that satisfies the invariant properties, the induction step can easily be finished. Due to the last invariant properties there is a proof for $\sigma'(\chi_i)$ for all premises χ_i of r. Now construct a new proof with as root $\sigma'(f(x_1,..,x_{r(f)}) \xrightarrow{a} t)$ and as direct subgraphs the proofs of the $\sigma'(\chi_i)$. Define $v' = \sigma'(t)$. Clearly, we have a proof for $f(v_1,..,v_{r(f)}) \xrightarrow{a} v'$. We may also conclude that for all $x \in Var(t)$: $\sigma(x) R \sigma'(x)$. By an application of the previously proved fact it follows that $\sigma(t) R \sigma'(t)$ or equivalently $u' R v'$. This completes the induction step except for the proof that a complete colouring exists.

In order to do this, it is sufficient to show that whenever we have a colouring which satisfies the invariant properties and in which only some nodes in Y are not coloured, we can extend this colouring with one element, while preserving the invariant. Let X' be such a colouring. We claim that there is some $i \in I$ such that t_i is coloured but y_i not. In order to see that this is true, assume that there is no such i. It cannot be that for some $j \in I$, y_j is coloured, but t_j is not coloured, because that would contradict with the assumption that all predecessors of a coloured node are coloured, too. Hence, we can partition I in two sets. $I_c = \{j \mid t_j \text{ and } y_j \text{ are coloured}\}$ and $I_{nc} = \{j \mid t_j \text{ and } y_j \text{ are not coloured}\}$. By assumption I_{nc} is non-empty. In the dependency graph G each element in I_{nc} has an incoming edge

from some element in I_{nc}. Hence, G contains a cycle and we have a contradiction. So let $i \in I$ with t_i coloured but y_i not coloured. We have that P proves the transition $\sigma(t_i) \xrightarrow{a_i} \sigma(y_i)$ with a proof of depth less than n and furthermore $\sigma(t_i) R\, \sigma'(t_i)$ because for all variables $x \in Var(t_i)$ $\sigma(x) R\, \sigma'(x)$ (use the previously proved fact). By definition of R we can distinguish between two cases:

1) $\sigma(t_i) \leftrightarrows_P \sigma'(t_i)$. In this case there is a $w \in T(\Sigma)$ such that $P \vdash \sigma'(t_i) \xrightarrow{a_i} w$ and $\sigma(y_i) R\, w$. This of course means that we can extend the definition of σ' to y_i by taking $\sigma'(y_i) = w$. One can easily check that the invariant properties of the colouring are preserved.
2) There is a function name g in F and there are terms w_j, w_j' for $1 \leqslant j \leqslant r(g)$ such that:

   $\sigma(t_j) = g(w_1,..,w_{r(g)})$,

   $\sigma'(t_j) = g(w_1',..,w_{r(g)}')$ and

   $w_j\, R\, w_j'$ for $1 \leqslant j \leqslant r(g)$.

   But now we can apply the induction hypothesis which gives that there is a w such that $P \vdash g(w_1',..,w_{r(g)}') \xrightarrow{a_i} w$ and $\sigma(y_i) R\, w$. Again we can extend the definition of σ' to y_i by taking $\sigma'(y_i) = w$ and it is easy to check that the invariant properties of the colouring are preserved.

This completes the proof of the induction step. □

5.13. The implication in theorem 5.10 cannot be reversed. The TSS $P'(\mathrm{BPA}_\delta)$ described in section 5.5 is not in *tyft/tyxt* format. But, as observed in that section, it is equivalent to the TSS $P(\mathrm{BPA}_\delta)$ which *is* in *tyft/tyxt* format. Hence, bisimulation equivalence is clearly a congruence. However, if one adds new operators and rules to $P(\mathrm{BPA}_\delta)$, then the congruence property can get lost, even if the rules for the new operators satisfy our *tyft/tyxt* format. In order to see this, consider the TSS obtained by adding to $P'(\mathrm{BPA}_\delta)$ *encapsulation* or *restriction* operators ∂_H for $H \subseteq Act$ and the *tyft* rules:

$$\frac{x \xrightarrow{a} x'}{\partial_H(x) \xrightarrow{a} \partial_H(x')} \quad a \notin H$$

We then obtain $a \leftrightarrows \partial_{\{b\}}(a)$, but $a \cdot b \not\leftrightarrows \partial_{\{b\}}(a) \cdot b$.

5.14. A different type of counterexample is given by a TSS P with constant names a,b and δ, a binary function name f, labels a,b and c and rules:

$$a \xrightarrow{a} \delta$$

$$b \xrightarrow{a} \delta$$

$$b \xrightarrow{b} \delta$$

$$f(a) \xrightarrow{c} \delta$$

The last rule is not in *tyft/tyxt* format, but it is not hard to see that $\leftrightarrows_P$ is a congruence. It is also not possible to write the above system as a TSS in *tyft/tyxt* format. If it were possible we would at least need a rule:

$$\frac{\cdots\cdots}{f(x) \xrightarrow{c} \delta}$$

But then we risk that $f(b) \xrightarrow{c} \delta$. We have to distinguish between a and b and we can only use the premises. But this attempt also fails as b can perform every action that can be performed by a. Again we have that adding *tyft/tyxt* rules may destroy the congruence property (take the axiom $a \xrightarrow{b} \delta$ or $c \xrightarrow{a} \delta$). Note that the example becomes less trivial if we add an extra constant name c and axioms $c \xrightarrow{a} \delta$ and $f(c) \xrightarrow{c} \delta$. Again, this constitutes a counterexample.

5.15. REMARK. The examples of sections 5.13 and 5.14 show that there is another reason for using TSS's in *tyft/tyxt*, namely their extensibility, without endangering congruence properties. It seems that, whenever a TSS contains a non *tyft/tyxt* rule, we can extend this TSS (except for some trivial cases) with a number of *tyft/tyxt* rules in such a way that for the resulting TSS bisimulation is not a congruence.

6. SOME APPLICATIONS

In this section we give some examples of TSS's and applications of the congruence theorem.

6.1. The silent move. In process algebra it is current practice to have a constant τ representing an internal machine step that cannot be observed. In order to describe the 'invisible' nature of τ, the notions of *observational congruence* [21] and *rooted-τ-bisimulation* [7] have been introduced. As observed by VAN GLABBEEK [13] it is not necessary to introduce a new notion of bisimulation if one uses TSS's. Below the signature $\Sigma(\mathrm{BPA}_{\delta}^{\tau})$ is enlarged with a constant name τ and rules are given that capture the notion of hidden, internal machine steps.
$P(\mathrm{BPA}_{\epsilon\delta}^{\tau})=(\Sigma(\mathrm{BPA}_{\epsilon\delta}^{\tau}),Act_{\tau\surd},R(\mathrm{BPA}_{\epsilon\delta}^{\tau}))$ with $Act_{\tau\surd}=Act_{\surd}\cup\{\tau\}$. $R(\mathrm{BPA}_{\epsilon\delta}^{\tau})$ consists of the combination of the rules in table 1 (but now a ranges over $Act_{\tau\surd}$) and table 2 (where a also ranges over $Act_{\tau\surd}$).

7.	$a \xrightarrow{a} \tau$	$a \neq \surd$
8.	$\dfrac{x \xrightarrow{a} y \quad y \xrightarrow{\tau} z}{x \xrightarrow{a} z}$	
9.	$\dfrac{x \xrightarrow{\tau} y \quad y \xrightarrow{a} z}{x \xrightarrow{a} z}$	

TABLE 2

An intuition one can have about a transition $\xrightarrow{a}$ $(a\neq\tau)$ is that one can see that a happens within a certain positive time interval. Then $\xrightarrow{\tau}$ means that no visible action can be observed during such an interval. Rule 8 and 9 fit naturally in this intuition. Rule 8 expresses that if one can observe a during a first time interval, and nothing in a subsequent second interval, then one observes a during the whole interval. For rule 9 one can have a same sort of intuition. For an action one can have the intuition that the execution of a process a takes a certain positive amount of time, but that observation of this process takes place at a particular moment, for instance at the beginning. Rule 7 then says that when we observe that a happens, it is still possible that some internal activity takes place before it terminates. The TSS $P(\mathrm{BPA}_{\epsilon\delta}^{\tau})$ is in *tyft/tyxt* format and (strong) bisimulation is a congruence. The theory $\mathrm{BPA}_{\epsilon\delta}^{\tau}$, given in table 3 with a ranging over elements from Act_{τ}, is a sound and complete axiomatisation of the semantics generated by the TSS $P(\mathrm{BPA}_{\epsilon\delta}^{\tau})$ modulo strong (!) bisimulation.
In figure 8-10 we give three examples corresponding to the τ-laws of MILNER [21]. In figure 8 two separate transition systems are drawn. In figure 8 and 10 a may not equal τ. In figure 9 the relevant states of $\tau+\epsilon$ and τ are drawn, as the equation $\tau+\epsilon=\tau$ is equivalent to the axiom T2. It is left to the reader to check that the expressions are strongly bisimilar.

$BPA^{\tau}_{\epsilon\delta}$				
	$x+y = y+x$	A1	$a\tau = a$	T1
	$x+(y+z) = (x+y)+z$	A2	$\tau x+x = \tau x$	T2
	$x+x = x$	A3	$a(\tau x+y) = a(\tau x+y)+ax$	T3
	$(x+y)z = xz+yz$	A4		
	$(xy)z = x(yz)$	A5		
	$x+\delta = x$	A6		
	$\delta x = \delta$	A7		
	$\epsilon x = x$	A8		
	$x\epsilon = x$	A9		

TABLE 3

6.2. Recursion. There are many ways to deal with recursion in process algebra. One approach is to introduce a set Ξ of *process names*. Elements of Ξ are added to the signature of the TSS as constants names. The recursive definitions of the process names are given by a set $E=\{X \Leftarrow t_X \mid X \in \Xi\}$ of *declarations*. Here the t_X are ground terms over the signature of the TSS (hence, they may contain process names in Ξ). If $X \Leftarrow t_X$ is a declaration, then this means that the behaviour of process X is given by its *body* t_X. Formally this is expressed by adding to the TSS rules:

$$\frac{t_X \xrightarrow{a} y}{X \xrightarrow{a} y}$$

for every declaration $X \Leftarrow t_X$. Now observe that these rules fit in the *tyft* format. Hence it follows that if one adds recursion to a TSS in *tyft/tyxt* format as described above, bisimulation remains a congruence.

A slightly different way of dealing with recursion is followed in [16,24]. Here axioms $X \xrightarrow{\tau} t_X$ appear saying that by some internal activity, a process name can expand to its body. Also this type of rules satisfy our format.

6.3. Consider the TSS $P(BPA^{\tau}_{\epsilon\delta})$ of example 6.1. In this system $\tau a \not\leftrightarrows a$ as one can easily see by remarking that a cannot perform an initial τ-transition. The inequality is generally defended by a congruence argument: 'If $\tau a \leftrightarrows a$ then by congruence one would expect that $b+\tau a \leftrightarrows b+a$, but this is intuitively not the case as only the left side can deadlock if action a is blocked but b not'. By adding the rule $x \xrightarrow{\tau} x'$ to $P(BPA^{\tau}_{\epsilon\delta})$, it follows that indeed $\tau a \leftrightarrows a$. By the congruence theorem it follows immediately that $b+\tau a \leftrightarrows b+a$.

6.4. The state operator. In many cases where operational semantics of a language is defined using Plotkin style rules, values play a role (see for instance [2,27]). Here, states of the transition system are generally configurations, i.e. pairs $<t,\sigma>$ of a process expression t and a valuation σ. In this section we argue that it is often possible to give inductive rules for these languages in a single sorted setting using the *extended state operator* Λ_σ of ACP [3].

We will define the state operator in the setting of $BPA^{\tau}_{\epsilon\delta}$ as given in section 6.1. Assume that S is a set of states. An expression $\Lambda_\sigma(t)$ with $\sigma \in S$ and t an expression in which state operators may occur, represents the process that transforms the state σ during successive transitions of t using a function $effect: S \times Act_\tau \times Act_\tau \to S$ while influencing the actual labels of the transitions of t as specified by a function $action: Act_\tau \times S \to 2^{Act_\tau}$. $action(a,\sigma)$ defines the set of actions that may be done in state σ by $\Lambda_\sigma(t)$ if t can perform a. $effect(\sigma,a,b)$ defines the resulting state if $\Lambda_\sigma(t)$ actually transforms under b while t performs a. Note that the extra argument b is necessary as the action function defines a set of possible actions that can be performed by $\Lambda_\sigma(t)$. The environment can determine which action from this set actually happens. The functions *effect* and *action* are *inert* for τ, i.e. $action(\tau,\sigma)=\{\tau\}$ and $effect(\sigma,\tau,a)=\sigma$ for every $a \in Act_\tau$. The inductive rules for the state operator are ($\sigma \in S$; $a,b \in Act_\tau$):

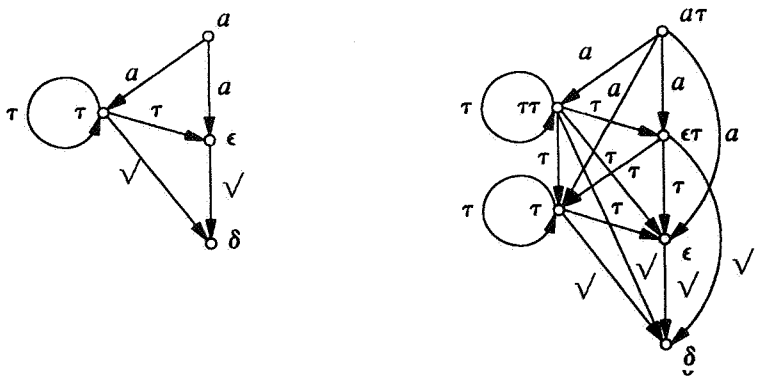


FIGURE 8 ($a = a\tau$)

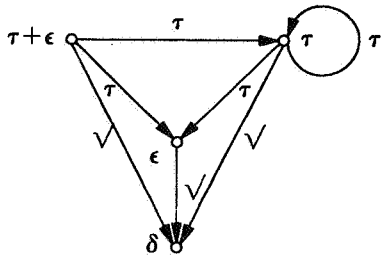


FIGURE 9 ($\tau+\epsilon = \tau$)

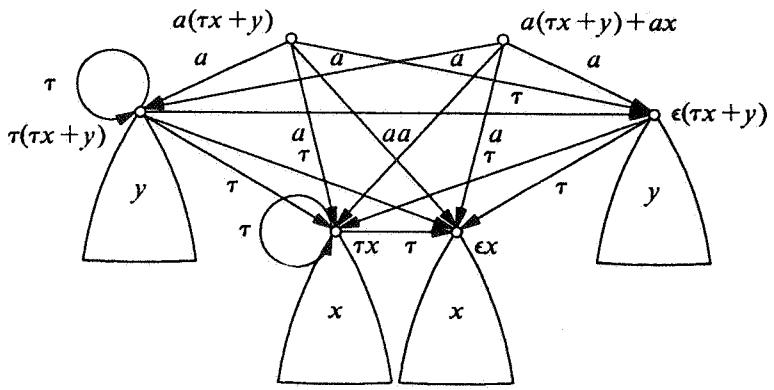


FIGURE 10 ($a(\tau x+y) = a(\tau x+y)+ax$)

$$\frac{x \xrightarrow{a} x'}{\Lambda_\sigma(x) \xrightarrow{b} \Lambda_{effect(\sigma,a,b)}(x')} \quad b \in action(a,\sigma),\ a \neq \surd$$

$$\frac{x \xrightarrow{\surd} x'}{\Lambda_\sigma(x) \xrightarrow{\surd} \Lambda_\sigma(x')}$$

As a typical example we consider a small subset of CSP. Actions in *Act* are of the form: $g!e$, $g?v$ or $[v:=e]$ with v from a set of variables $\mathcal{V}$ and e a natural number expression which may contain variables from $\mathcal{V}$ and natural numbers with their usual operations such as $+,-,\times$. $g!e$ means 'write the value of expression e to channel g', $g?v$ means 'read a value from channel g and assign this value to variable v' and $[v:=e]$ means: 'assign the value of expression e to v'. We assume that we have an interpretation function $[\![\cdot]\!]$ that, given a valuation σ of the variables, yields for each expression a

natural number. We take as state space S all valuations $\sigma:\mathcal{V}\to\mathbb{N}$. Let $\sigma[n/v]$ be the valuation σ except for the fact that variable v is mapped on n. Now we can define the function *action* and *effect* as follows:

$$\begin{array}{ll} action(\sigma, g!e) = \{g![\![e]\!]^\sigma\} & effect(\sigma, g!e, g!n) = \sigma \\ action(\sigma, g?v) = \{g?n \mid n\in\mathbb{N}\} & effect(\sigma, g?v, g?n) = \sigma[n/v] \\ action(\sigma, [v:=e]) = \{\tau\} & effect(\sigma, [v:=e], \tau) = \sigma[[\![e]\!]^\sigma/v] \end{array}$$

The function *effect* is inert in the remaining cases. Now consider a process that is capable of reading a value from channel g_1 and sending its square to channel g_2:

$$\Lambda_\sigma(g_1?v\cdot[w:=v\times v]\cdot g_2!w)$$

A particular sequence of transitions that can be performed by this process is:

$$\Lambda_\sigma(g_1?v\cdot[w:=v\times v]\cdot g_2!w)\xrightarrow{g_1?3}\Lambda_{\sigma[3/v]}(\epsilon\cdot[w:=v\times v]\cdot g_2!w)\xrightarrow{\tau}$$
$$\Lambda_{\sigma[3/v,9/w]}(\epsilon\cdot g_2!w)\xrightarrow{g_2!9}\Lambda_{\sigma[3/v,9/w]}(\epsilon)\xrightarrow{\surd}\Lambda_{\sigma[3/v,9/w]}(\delta)$$

It is not hard to see that $BPA^\tau_{\epsilon\delta}$ with the state operator can be extended with a parallel combinator, for instance the parallel operator from ACP. Then, communication can be defined such that we have value passing between several processes. We will not give a detailed elaboration of this as it is beyond the scope of this article. However, we would like to stress that some sense the extended state operator is more powerful than the approach with a global state using configurations. The extended state operator can in a very natural way be used to model that certain data are local to some processes.

7. Modular properties of transition system specifications

A very natural operation on TSS's is to take their componentwise union. Given two TSS's P_0 and P_1 we use the notation $P_0\oplus P_1$ to denote the resulting system. A nice property to have in such a situation is that the outgoing transitions in $TS(P_0)$ of terms in the signature of P_0 are the same as the outgoing transitions of these terms in $TS(P_0\oplus P_1)$. This means that $P_0\oplus P_1$ is a *conservative extension* of P_0: any property which has been proved for the states in the old transition system remains valid (for the old states) in the enriched system.
In this section we study the question what restrictions we have to impose on P_0 and P_1 in order to obtain conservativity. First we give the basic definitions.

7.1. Definition. Let $\Sigma_i=(F_i,r_i)$ $(i=0,1)$ be two signatures such that $f\in F_0\cap F_1 \Rightarrow r_0(f)=r_1(f)$. The *sum* of Σ_0 and Σ_1, notation $\Sigma_0\oplus\Sigma_1$, is the signature:

$$\Sigma_0\oplus\Sigma_1=(\Sigma_0\cup\Sigma_1,\lambda f.\textit{if } f\in F_0 \textit{ then } r_0(f) \textit{ else } r_1(f)).$$

7.2. Definition. Let $P_i=(\Sigma_i,A_i,R_i)$ $(i=0,1)$ be two TSS's with $\Sigma_0\oplus\Sigma_1$ defined. The *sum* of P_0 and P_1, notation $P_0\oplus P_1$, is the TSS:

$$P_0\oplus P_1=(\Sigma_0\oplus\Sigma_1,A_0\cup A_1,R_0\cup R_1).$$

7.3. Definition. Let $P_i=(\Sigma_i,A_i,R_i)$ $(i=0,1)$ be two TSS's with $P=P_0\oplus P_1$ defined. Let $P=(\Sigma,A,R)$. We say that P is a *conservative extension* of P_0 and that P_1 *can be added conservatively* to P_0 if for all $s\in T(\Sigma_0)$, $a\in A$ and $t\in T(\Sigma)$:

$$P\vdash s\xrightarrow{a}t \;\Leftrightarrow\; P_0\vdash s\xrightarrow{a}t.$$

Note that the implication $P\vdash s\xrightarrow{a}t \Leftarrow P_0\vdash s\xrightarrow{a}t$ holds trivially. The following example illustrates

the use of conservativity:

7.4. EXAMPLE. Let $P_i=(\Sigma_i,A_i,R_i)$ $(i=0,1)$ be two TSS's with $P=P_0\oplus P_1$ a conservative extension of P_0. Then P is also a conservative extension of P_0 up to bisimulation, i.e. for $s,t\in T(\Sigma_0)$:

$$s\leftrightarrows_P t \quad\Leftrightarrow\quad s\leftrightarrows_{P_0} t.$$

7.5. COUNTEREXAMPLES. We want to study the question in which cases a TSS P_1 can be added conservatively to a TSS P_0. However, we will restrict ourselves to the case where both P_0 and P_1 are in *tyft/tyxt* format. Below, 5 examples are presented that illustrate situations where we do not have conservativity.

7.5.1. EXAMPLE. If P_1 has a rule with a function name that already occurred in Σ_0 in the lhs of the conclusion, then problems arise quite soon. If $P_0=P(\mathrm{BPA}_\delta)$ and P_1 contains a single rule:

$$x+y\xrightarrow{ko}\delta$$

then $\epsilon\leftrightarrows_{P_0}\epsilon+\epsilon$ but not $\epsilon\leftrightarrows_{P_0\oplus P_1}\epsilon+\epsilon$.

7.5.2. EXAMPLE. Conservativity can get lost if free variables occur in a premise of a rule in P_0. In order to see this consider the TSS P_0 with unary function name $a:$, constants b, c, a label a and rules:

$$a{:}x\xrightarrow{a}x$$

$$b\xrightarrow{a}b$$

$$\frac{x\xrightarrow{a}y}{c\xrightarrow{a}y}$$

It is not hard to see that $b\leftrightarrows c$. However, if we add a constant name d, a label d and rule $d\xrightarrow{d}d$ it follows that $b\not\leftrightarrows c$.

7.5.3. EXAMPLE. Conservativity can get lost also if free variables occur in the conclusion of a rule in P_0. Suppose the signature of P_0 consists of a constant symbol a and a unary function name f. The set of labels consists of a and there are two axioms:

$$a\xrightarrow{a}a$$

$$f(x)\xrightarrow{a}y$$

It is not hard to see that $a\leftrightarrows_{P_0} f(a)$. However, if we add a TSS P_1 which contains an axiom $b\xrightarrow{b}b$, then $a\not\leftrightarrows_{P_0\oplus P_1} f(a)$.

7.5.4. EXAMPLE. Conservativity can be violated if we add *tyxt* rules to a given TSS P_0 as it allows us to add new transitions to states in the transition system $TS(P_0)$. It even threatens conservativity up to bisimulation if we add *tyxt* rules with labels in the conclusion that already occurred in A_0. Let P_0 consist of $P(\mathrm{BPA}_\delta)$ together with the rule:

$$\frac{x\xrightarrow{ok}x'}{x+y\xrightarrow{ko}x'}$$

In P_0 we have $\epsilon\leftrightarrows\epsilon+\epsilon$. This is no longer true if we add a TSS which contains a single axiom $x\xrightarrow{ok}x$.
Another example of this kind is given by the rules 8 and 9 in table 2. Consider $P(\mathrm{BPA}_\delta)$ to which rule 7 has been added. None of the τ-laws hold in this system. If rule 8 and 9 are added they do. Hence, these rules do not preserve conservativity up to bisimulation.

7.5.5. EXAMPLE. Our last example shows that circularities in P_0 can disturb conservativity. Suppose P_0 consists of $P(\mathrm{BPA}_\delta^*)$ and a circular rule:

$$\frac{x_1+y_1 \xrightarrow{ok} y_2 \quad x_2+y_2 \xrightarrow{ok} y_1}{x_1+x_2 \xrightarrow{ko} y_1+y_2}$$

It is not hard to see that $\epsilon \leftrightarrows_{P_0} \epsilon+\epsilon$. However, adding a TSS P_1 with a single axiom $ok \xrightarrow{ok} ok$ makes that $\epsilon \not\leftrightarrows_{P_0 \oplus P_1} \epsilon+\epsilon$.

The next theorem shows that the examples above give a complete overview of the situations in which we do not have conservativity.

7.6. THEOREM. *Let $P_0=(\Sigma_0,A_0,R_0)$ be a TSS in pure tyft/tyxt format and let $P_1=(\Sigma_1,A_1,R_1)$ be a TSS in tyft format such that there is no rule in R_1 that contains a function name from Σ_0 in the left hand side of its conclusion. Let $P=P_0 \oplus P_1$ be defined. Then P_1 can be added conservatively to P_0.*

PROOF. Let $P=(\Sigma,A,R)$. Let $s \in T(\Sigma_0)$, $a \in A$ and $s' \in T(\Sigma)$ with $P \vdash s \xrightarrow{a} s'$. Let T be a proof of $s \xrightarrow{a} s'$ from P. With induction on the size of T we prove that T is also a proof of $s \xrightarrow{a} s'$ from P_0. Let r be the last rule which is used in T. Because $s \in T(\Sigma_0)$ and all rules of P_1 are in *tyft* format and contain no function names from Σ_0 in the left hand side of their conclusions, r must be a rule in P_0. Suppose r is in pure *tyft* format (the case that r is in pure *tyxt* format is completely analogous and omitted). Suppose in particular that r is equal to:

$$\frac{\{t_i \xrightarrow{a_i} y_i \mid i \in I\}}{f(x_1,..,x_{r(f)}) \xrightarrow{a} t}$$

Let σ be the substitution that relates rule r to the last step in proof T and let $\{s_i \xrightarrow{a_i} s_i' \mid i \in I\}$ be the set of labels of nodes directly above the root of T. We then have:

$$\sigma(t_i)=s_i,$$
$$\sigma(y_i)=s_i',$$
$$\sigma(f(x_1,..,x_{r(f)})=s,$$
$$\sigma(t)=s'.$$

Now we use the same type of strategy as in the proof of theorem 5.10. Call a variable x in $Var(r)$ *coloured* as soon as it is shown that $\sigma(x)$ is in $T(\Sigma_0)$. Because $s \in T(\Sigma_0)$ and $\sigma(f(x_1,..,x_{r(f)})=s$, all variables $x_1,..,x_n$ are coloured. Now if r does not have a premise containing a variable that is not coloured, we are ready: by induction the proofs of transitions $s_i \xrightarrow{a_i} s_i'$ are proofs from P_0 and since t contains no free variables $\sigma(t)=s'$ is in $T(\Sigma_0)$. Hence T is a proof from P_0. If r contains a premise with a variable that is not coloured, then, due to the fact that r is non circular and contains no free variables, there must be a premise $t_i \xrightarrow{a_i} y_i$ with all variables in t_i coloured but y_i not coloured. But in that case we can apply induction: since $s_i=\sigma(t_i) \in T(\Sigma_0)$, $s_i'=\sigma(y_i) \in T(\Sigma_0)$ too, because $P \vdash s_i \xrightarrow{a_i} s_i'$ with a proof smaller than T. Thus we have a coloured y_i. This argument can be iterated until all variables in $Var(r)$ are coloured. □

7.7. In our view the counterexamples which show that the original system has to be pure and no rule from the added system may contain a function symbol from the original system in the lhs of its conclusion are quite strong. It will be very difficult to strengthen theorem 7.6 by weakening these constraints, perhaps with exception of the constraints about circularity. Because modularity is an important and desirable property and because TSS's which are not pure are ill-behaved with respect to modularisation, one might decide, for this reason, to call such TSS's unstructured.
The main reason we had for including theorem 7.6 in this paper is that we need it in the next section.

It is clear that a lot more can be said about modular properties of TSS's than we have done here. Especially, if one is satisfied with conservativity up to bisimulation, it seems possible to include (under certain conditions) *tyxt* rules in the extending TSS. However, we want to leave this as a topic for future research.

8. Completed trace congruence

In this section we study the completed trace congruence induced by the pure *tyft/tyxt* format. Intuitively, two processes s and t are completed trace congruent if for any context $C[\,]$ which can be defined using the pure *tyft/tyxt* format, the completed traces of $C[s]$ and $C[t]$ are the same. It seems reasonable to require that, whenever new function names and rules are added to a TSS in order to build a context which can distinguish between terms, these new ingredients may not change the original transition system: the extension should be conservative. If it would be allowed to introduce new transitions in the original transition system, then we could add rules like:

$$\frac{x \xrightarrow{I'm(s)} x',\ y \xrightarrow{I'm(t)} y'}{x+y \xrightarrow{I'm(s+t)} x'+y'}$$

and make that syntactically different terms always have outgoing transitions with different labels. As a result completed trace congruence would just be syntactic equality between terms.
The results of the previous section show that for a TSS in *tyft/tyxt* format it is in general rather difficult to determine a class of TSS's which can be added to it conservatively. Consequently it is also difficult to characterize the completed trace congruence induced by this format. However, for TSS's in pure *tyft/tyxt* format such a class exists: by theorem 7.6 every TSS in *tyft* format can be added conservatively to a TSS in pure *tyft/tyxt* format. For this reason we decided to work on a characterization of the completed trace congruence induced by the pure *tyft/tyxt* format and leave the general *tyft/tyxt* format for what it is. We think that this is not a serious restriction because:

- We have never seen an application of a TSS with circular rules or rules with free variables.
- Absence of circularities is used anyhow in the proof of theorem 5.10. The proof of lemma 5.9 shows that for every non circular TSS in *tyft/tyxt* format there exists an equivalent TSS in pure *tyft/tyxt* format.
- TSS's in *tyft/tyxt* format that are not pure, are ill-behaved with respect to modularisation and therefore not much effort should be spent in proving theorems about them.

8.1. Definition. Let $\mathcal{A}=(S,A,\rightarrow)$ be a LTS and let $s\in S$. s is a *termination node*, notation $s\not\rightarrow$, if there are no $t\in S$ and $a\in A$ with $s\xrightarrow{a}t$. A sequence $\sigma\in A^*$ is a *completed trace* of s if there are actions $a_1,a_2,..,a_n\in A$ with $\sigma=a_1{*}a_2{*}..{*}a_n$ and states $s_1,s_2,..,s_n\in S$ such that $s\xrightarrow{a_1}s_1\xrightarrow{a_2}..\xrightarrow{a_n}s_n\not\rightarrow$. $CT(s)$ is the set of all completed traces of s. Two states $s,t\in S$ are *completed trace equivalent* if $CT(s)=CT(t)$. This is denoted as $s\equiv_{CT}t$.

8.2. Definition. Let $\mathcal{F}$ be some format of TSS rules. Let $P=(\Sigma,A,R)$ be a TSS in $\mathcal{F}$ format. Two terms $s,t\in T(\Sigma)$ are *completed trace congruent with respect to* $\mathcal{F}$ rules, notation $s\equiv_{\mathcal{F}}t$, if for every TSS $P'=(\Sigma',A',R')$ in $\mathcal{F}$ format which can be added conservatively to P and for every $\Sigma\oplus\Sigma'$-context $C[\,]$: $C[s]\equiv_{CT}C[t]$. s and t are *completed trace congruent within* P, notation $s\equiv_P t$, if for every Σ-context $C[\,]$: $C[s]\equiv_{CT}C[t]$.

8.3. Note. In the sequel we will define a number of equivalence relations on the states of transition systems. If $P=(\Sigma,A,R)$ is a TSS and s,t are terms in $T(\Sigma)$ then, whenever we say that s and t are equivalent according to a certain equivalence relation, what we mean is that the states s and t of the transition system $TS(P)$ are equivalent according to this relation.

8.4. ABRAMSKY [1] and BLOOM, ISTRAIL & MEYER [10] give Plotkin style rules to define operators with which one can distinguish between any pair of non-bisimilar processes. We cannot obtain this result with pure *tyft/tyxt* rules, but we will show that the notion of completed trace congruence with respect to pure *tyft/tyxt* rules exactly coincides with *2-nested simulation equivalence* for all *image finite* processes. What we in fact will prove is best illustrated by figure 11.

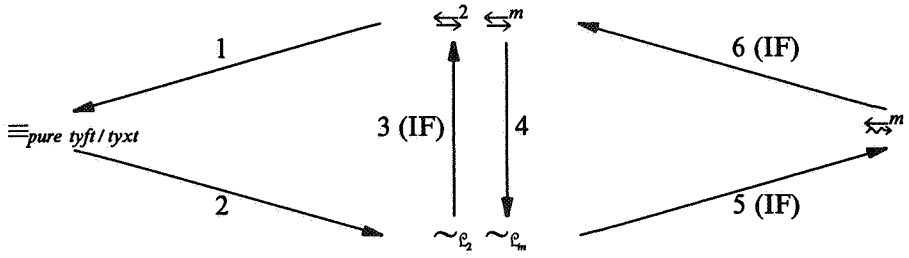


Figure 11

The arrows indicate set inclusion. 'IF' stands for Image Finite and indicates that we need image finiteness of processes for the proofs of inclusions 3,5 and 6. $\leftrightarrows^m$ is m-nested simulation equivalence. $\sim_{\mathcal{L}_m}$ is the equivalence induced by the set $\mathcal{L}_m$ of Hennessy-Milner formulas in which no negation symbol $\neg$ occurs nested m times or more. In the right corner of figure 11 we have an auxiliary equivalence notion $\leftrightarrows^m$. In the rest of this section these notions are made precise and the inclusions are proved. It immediately follows that both triangles collapse for image finite transitions systems. In particular we have the following theorem.

8.4.1. DEFINITION. Let $\mathcal{A}=(S,A,\rightarrow)$ be a LTS. $\mathcal{A}$ is called *image finite* if for all $s\in S$ and $a\in A$ the set $\{t \mid s \xrightarrow{a} t\}$ is finite.

8.4.2. THEOREM. *Let $P=(\Sigma,A,R)$ be a TSS in pure tyft/tyxt format such that $TS(P)$ is image finite. Let $s,t\in T(\Sigma)$. Then:*

$$s \equiv_{pure\ tyft/tyxt} t \quad \Leftrightarrow \quad s \leftrightarrows^2 t \quad \Leftrightarrow \quad s \sim_{\mathcal{L}_2} t.$$

8.4.3. Bloom, Istrail & Meyer have studied the completed trace congruence induced by 'tree rules'. Tree rules differ from pure *tyft/tyxt* rules in that they may only have variables in the premises and there may not be a single variable in the left hand side of a conclusion. Hence, one could also call this type of rules 'pure *xyft* rules'. They proved the following theorem [9]:

8.4.4. THEOREM (BLOOM, ISTRAIL & MEYER). *Let $P=(\Sigma,A,R)$ be a TSS in tree rule format such that $TS(P)$ is image finite. Let $s,t\in T(\Sigma)$. Then:*

$$s \equiv_{tree\ rules} t \quad \Leftrightarrow \quad s \sim_{\mathcal{L}_2} t.$$

This result, which is close to our characterization theorem, has not been published. A sketch of the proof is included at the end of this section. We were aware of the result of Bloom, Istrail & Meyer before we proved the characterization theorem for the pure *tyft/tyxt* format. However, all proofs in this section are entirely our own.

8.5. Here definitions are given of m-nested simulation equivalence ($\leftrightarrows^m$). Further inclusion 1 is proved.

8.5.1. DEFINITION. Let $\mathcal{A}=(S,A,\rightarrow)$ be a LTS. A relation $R \subseteq S \times S$ is called a *simulation* if it satisfies:
- whenever $s\,R\,t$ and $s \xrightarrow{a} s'$ then, for some $t' \in S$, also $t \xrightarrow{a} t'$ and $s'R\,t'$.

s can be simulated by t, notation $s \leftrightarrows t$, if there is a simulation containing the pair (s,t). s and t are *simulation equivalent*, notation $s \leftrightarrows t$, if $s \leftrightarrows t$ and $t \leftrightarrows s$.

8.5.2. DEFINITION. Let $\mathcal{A}=(S,A,\rightarrow)$ be a LTS. We define a sequence of relations $\leftrightarrows^m$ ($m \geqslant 0$) as follows:
i) $\leftrightarrows^0 = S \times S$,
ii) A relation $R \subseteq S \times S$ is an *$m+1$-nested simulation* if it is a simulation contained in $(\leftrightarrows^m)^{-1}$. State s *can be simulated $m+1$-nested by* state t, notation $s \leftrightarrows^{m+1} t$, if there exists an $m+1$-nested simulation containing the pair (s,t).

Two states s and t are *m-nested simulation equivalent*, notation $s \leftrightarrows^m t$ if $s \leftrightarrows^m t$ and $t \leftrightarrows^m s$.

Observe that 1-nested simulation equivalence is the same as simulation equivalence.

8.5.3. LEMMA. *For $m \geqslant 0$, $\leftrightarrows^{m+1} \subseteq \leftrightarrows^{m+1} \subseteq \leftrightarrows^m$.*
PROOF. Straightforward using the definitions. □

8.5.4. EXAMPLE. For every $m \geqslant 0$ we can find processes that are m-nested similar, but not $m+1$-nested similar. Consider the TSS $P(\mathrm{BPA}_\delta)$. Let the processes s_m, t_m be defined for each $m \geqslant 0$ as follows:

$$\begin{aligned} s_0 &= c\delta & t_0 &= c\delta + b\delta \\ s_{m+1} &= at_m & t_{m+1} &= as_m + at_m \end{aligned}$$

Below in figure 12 a part of the transition system is displayed (some ϵ's are dropped):

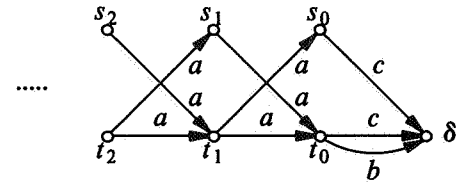


FIGURE 12

One can easily prove that: $s_m \leftrightarrows^m t_m$ and $s_m \not\leftrightarrows^{m+1} t_m$ for each $m \geqslant 0$.

8.5.5. EXAMPLE. It is well known that simulation equivalence does not respect completed trace congruence. Take for instance the processes s_1 and t_1 of the previous example. Now put them in the context $\partial_{\{c\}}([\,])$ where $\partial_{\{c\}}$ is the encapsulation operator as defined in section 5.13. The completed traces of these processes clearly differ: $CT(\partial_{\{c\}}(a(c\delta+b\delta))) = \{a \star b\}$ and $CT(\partial_{\{c\}}(ac\delta+a(c\delta+b\delta))) = \{a, a \star b\}$. In fact simulation equivalence doesn't even refine completed trace equivalence. Take for example the simulation equivalent processes a and $a\delta+a$. The maximal trace sets are $\{a \star \surd\}$ and $\{a, a \star \surd\}$, respectively. However, it is not hard to see that for $m \geqslant 2$, m-nested simulation equivalence refines completed trace equivalence.

8.5.6. LEMMA. *Let $\Sigma=(F,r)$ be a signature and let $P=(\Sigma,A,R)$ be a TSS. If P is non circular and in tyft/tyxt format, then $\leftrightarrows^m$ $(m\geqslant 0)$ is a congruence for all function names in F.*
PROOF. Completely analogous to the proof of theorem 5.10 and left to the reader. □

The next theorem states the validity of inclusion 1.

8.5.7. THEOREM *(inclusion 1). Let $P=(\Sigma,A,R)$ be a TSS that is in pure tyft/tyxt format. Then:*

$$\leftrightarrows^2 \subseteq \equiv_{\text{pure tyft/tyxt}}.$$

PROOF. Let $s,t\in T(\Sigma)$ with $s\leftrightarrows^2 t$. Let $P'=(\Sigma',A',R')$ be a TSS in pure *tyft/tyxt* format that can be added conservatively to P and let $C[\,]$ be a $\Sigma\oplus\Sigma'$-context. Since $P\oplus P'$ is a conservative extension of P, $s\leftrightarrows^2 t$ in $TS(P\oplus P')$. Now we use that $\leftrightarrows^2$ is a congruence for operators in pure *tyft/tyxt* format (lemma 8.5.6) and get $C[s]\leftrightarrows^2 C[t]$. Since $\leftrightarrows^2$ refines completed trace congruence: $C[s]\equiv_{CT} C[t]$. Because P' and $C[\,]$ were chosen arbitrarily this gives us: $s\equiv_{\text{pure tyft/tyxt}} t$. □

8.6. Hennessy-Milner logic. Next we give the definitions of Hennessy-Milner logic (HML) and prove the second inclusion in figure 11. Most definitions are standard and can also be found in [17]. The notion of HML-formulas of alternation depth m seems to be new, although the set of HML-formulas of alternation depth 1 (the formulas without negation) is exactly the set $\mathfrak{M}$ of [17].

8.6.1. DEFINITION. The set $\mathfrak{L}$ of *Hennessy-Milner logic (HML) formulas* (over a given alphabet $A=\{a,b,\ldots\}$) is given by the following grammar:

$$\phi ::= T \mid \phi\wedge\phi \mid \neg\phi \mid \langle a\rangle\phi.$$

Let $\mathcal{L}=(S,A,\rightarrow)$ be a LTS. The satisfaction relation $\models \subseteq S\times\mathfrak{L}$ is the least relation such that:
- $s\models T$ for all $s\in S$,
- $s\models\phi\wedge\psi$ iff $s\models\phi$ and $s\models\psi$,
- $s\models\neg\phi$ iff not $s\models\phi$,
- $s\models\langle a\rangle\phi$ iff for some $t\in S$: $s\xrightarrow{a}t$ and $t\models\phi$.

We adopt the following notations:
- F stands for $\neg T$,
- $\phi\vee\psi$ stands for $\neg(\neg\phi\wedge\neg\psi)$,
- $[a]\phi$ stands for $\neg\langle a\rangle\neg\phi$.

It is not difficult to see that any HML formula is logically equivalent to a formula in the language $\mathfrak{L}'$ which is generated by the following grammar:

$$\phi ::= T \mid F \mid \phi\wedge\phi \mid \phi\vee\phi \mid \langle a\rangle\phi \mid [a]\phi.$$

8.6.2. DEFINITION. Let $\mathcal{L}=(S,A,\rightarrow)$ be a LTS and let $\mathfrak{K}$ be a set of HML formulas. With $\sim_{\mathfrak{K}}$ we denote the equivalence relation on S induced by $\mathfrak{K}$:

$$s\sim_{\mathfrak{K}} t \quad\Leftrightarrow\quad (\forall\phi\in\mathfrak{K}:\ s\models\phi \ \Leftrightarrow\ t\models\phi).$$

We will call this relation $\mathfrak{K}$ *formula equivalence.*

We recall a fundamental result of [17]:

8.6.3. THEOREM (HENNESSY & MILNER). *Let $\mathcal{A}=(S,A,\rightarrow)$ be an image finite LTS. Then for all $s,t\in S$:*

$$s \leftrightarrows t \quad \Leftrightarrow \quad s \sim_{\mathcal{L}} t.$$

8.6.4. DEFINITION. For $m\in\mathbb{N}$ we define the set $\mathcal{L}_m$ of HML-formulas by:
- $\mathcal{L}_0$ is empty,
- $\mathcal{L}_{m+1}$ is given by the following grammar:

$$\phi ::= \neg\psi (\text{for } \psi\in\mathcal{L}_m) \mid T \mid \phi\wedge\phi \mid \langle a\rangle\phi.$$

We leave it as an exercise to the reader to check that the equivalence induced by $\mathcal{L}_m$ formulas is the same as the equivalences induced by the sets $\mathcal{K}^{\langle\rangle}_m$ and $\mathcal{K}^{[]}_m$ which are given by:
- $\mathcal{K}^{\langle\rangle}_0 = \mathcal{K}^{[]}_0 = \varnothing$.
- $\mathcal{K}^{\langle\rangle}_{m+1}$ is defined by:

$$\phi ::= \psi (\text{for } \psi\in\mathcal{K}^{[]}_m) \mid T \mid F \mid \phi\wedge\phi \mid \phi\vee\phi \mid \langle a\rangle\phi.$$

- $\mathcal{K}^{[]}_{m+1}$ is defined by:

$$\phi ::= \psi (\text{for } \psi\in\mathcal{K}^{\langle\rangle}_m) \mid T \mid F \mid \phi\wedge\phi \mid \phi\vee\phi \mid [a]\phi.$$

8.6.5. EXAMPLE. Consider example 8.5.4. Define for $m\geq 1$ the formula $\varphi_m\in\mathcal{L}_m$ by: $\varphi_1=\langle b\rangle T\wedge\langle c\rangle T$ and $\varphi_{m+1}=\langle a\rangle\neg\varphi_m$. It is easily checked that for $i\geq 0$: $s_i\not\vDash\varphi_{i+1}$ and $t_i\vDash\varphi_{i+1}$.

8.6.6. THEOREM *(Testing $\mathcal{L}_2$ formulas). Let $P_0=(\Sigma_0,A_0,R_0)$ be a TSS in pure tyft/tyxt format. Then there is a TSS $P_1=(\Sigma_1,A_1,R_1)$ in pure tyft format, which can be added conservatively to P_0, such that completed trace congruence within $P_0\oplus P_1$ is included in $\mathcal{L}_2$ formula equivalence.*

PROOF. P_1 is constructed in the following way. The set A_1 consists of A_0 together with 5 new labels:

$$A_1 = A_0 \cup \{ok, left, right, syn, skip\}.$$

Signature Σ_1 contains a constant δ, unary function names $a:$ for each $a\in A_1$, and binary function names $+$ and *Sat*. Remark that the signature is finite if the alphabet A_0 is finite. For δ and $+$ we have just the same rules as in $BPA^{\S}_{\delta}$ and $a:$ denotes prefixing like in example 5.11.3. The most interesting operator is the operator *Sat*. Its first argument is intended to be a coding of some $\mathcal{L}_2$ formula. The *Sat* operator tests whether its second argument satisfies the $\mathcal{L}_2$ formula which is represented by its first argument. The rules of the *Sat* operator are given in table 4. In the table a ranges over A_1. Because P_1 is in *tyft* format, $\Sigma_0\cap\Sigma_1=\varnothing$ and P_0 is pure *tyft/tyxt*, it follows with theorem 7.6 that P_1 is a conservative extension of P_0.
Every formula is encoded using the following rules:

$$C_T = skip:\delta,$$
$$C_{\phi\wedge\psi} = left:C_\phi + right:C_\psi,$$
$$C_{\neg\phi} = skip:C_\phi,$$
$$C_{\langle a\rangle\phi} = syn:a:C_\phi.$$

We claim that for $\phi\in\mathcal{L}_2$, $Sat(C_\phi,t)$ has a completed trace *ok* iff $t\vDash\phi$. With this claim we can finish the proof: whenever for some $s,t\in T(\Sigma_0\oplus\Sigma_1)$ with $s\not\sim_{\mathcal{L}_2} t$, then there is an $\mathcal{L}_2$ formula ϕ_0 such that $s\vDash\phi_0$ and $t\not\vDash\phi_0$ (or vice versa). Using the claim this means that $Sat(C_{\phi_0},s)\not\equiv_{CT} Sat(C_{\phi_0},t)$.
Now we give some intuition about how $Sat(C_\phi,t)$ tests the formula ϕ on t. If we take $\phi=T$, testing is straightforward: $C_T=skip:\delta$ and *skip* indicates to *Sat* that it can do an *ok* step (rule 1). Hence,

$\dfrac{x \xrightarrow{skip} x'}{Sat(x,y) \xrightarrow{ok} Sat(x',y)}$	1
$\dfrac{x \xrightarrow{left} x_l,\ Sat(x_l,y) \xrightarrow{ok} y_l \quad x \xrightarrow{right} x_r,\ Sat(x_r,y) \xrightarrow{ok} y_r}{Sat(x,y) \xrightarrow{ok} y_l + y_r}$	2
$\dfrac{x \xrightarrow{syn} x',\ x' \xrightarrow{a} x'' \quad y \xrightarrow{a} y',\ Sat(x'',y') \xrightarrow{ok} y''}{Sat(x,y) \xrightarrow{ok} y''}$	3

TABLE 4

$Sat(skip:\delta,t) \xrightarrow{ok} Sat(\delta,t)$ and it is not hard to check that $Sat(\delta,t)$ cannot do a next step.
Testing of $\wedge$ and $\langle a \rangle$ is almost as straightforward as testing the formula T and resembles the definition of $\models$. The intuitive meaning of the constant names *left:*, *right:* and *syn:* is respectively: transform to the left/right part of a formula and synchronize the next action of the coded formula and the tested process. Testing $\neg$ contains a little trick. First, the positive part of a formula is tested, which possibly yields a first *ok* and then the negative parts are tested. This can give rise to another *ok*. For instance the test $Sat(C_{\neg\phi},t)$ performs an initial *ok* step as its positive part is empty and then tests for the $\mathcal{L}_1$ formula ϕ whether $t \models \phi$. If there is no negative part that holds, the test does not yield another *ok* action and there is a completed trace *ok*. If a negative part is true, the test will yield another *ok* step and the *ok* trace is extended to the trace $ok \star ok$, which is not ok because now $ok \notin MT(Sat(C_\phi,t))$. Next we will give, in two lemmas, a formal proof of the claim.

8.6.6.1. LEMMA. *Let* $t \in T(\Sigma_0 \oplus \Sigma_1)$ *and let* $\phi \in \mathcal{L}_1$. *Then:*
i) $t \models \phi \Rightarrow CT(Sat(C_\phi,t)) = \{ok\}$,
ii) $t \not\models \phi \Rightarrow CT(Sat(C_\phi,t)) = \varnothing$.

PROOF. Induction on the structure of ϕ.
a) ϕ is T. Then $t \models \phi$. The only move of $Sat(C_\phi,t)$ is $Sat(C_\phi,t) \xrightarrow{ok} Sat(\delta,t) \not\rightarrow$. Both implications hold.
b) ϕ is $\phi_1 \wedge \phi_2$. If $t \models \phi$ then $t \models \phi_1$ and $t \models \phi_2$. By induction $CT(Sat(C_{\phi_1},t)) = \{ok\}$ and $CT(Sat(C_{\phi_2},t)) = \{ok\}$. Since all outgoing transitions of $Sat(C_\phi,t)$ are proved using rule 2 in table 4, one can easily see that $CT(Sat(C_\phi,t)) = \{ok\}$. If on the other hand $t \not\models \phi$ then either $t \not\models \phi_1$ or $t \not\models \phi_2$. Hence by induction either $CT(Sat(C_{\phi_1},t)) = \varnothing$ or $CT(Sat(C_{\phi_2},t)) = \varnothing$, thus $Sat(C_\phi,t)$ can have no outgoing transitions and $CT(Sat(C_\phi,t)) = \varnothing$.
c) ϕ is $\langle a \rangle \phi'$. If $t \models \phi$ then there is a t' such that $t \xrightarrow{a} t'$ and $t' \models \phi'$. By induction $CT(Sat(C_{\phi'},t')) = \{ok\}$. Outgoing transitions of $Sat(C_\phi,t)$ can only be proved using rule 3 and inspection of this rule allows us to conclude that $CT(Sat(C_\phi,t)) = \{ok\}$. If $t \not\models \phi$ then for all t' with $t \xrightarrow{a} t'$, $t' \not\models \phi'$. Hence by induction $CT(Sat(C_{\phi'},t')) = \varnothing$. But this implies $CT(Sat(C_\phi,t)) = \varnothing$ since rule 3 cannot be applied. □

8.6.6.2. LEMMA. *Let* $t \in T(\Sigma_0 \oplus \Sigma_1)$ *and let* $\phi \in \mathcal{L}_2$. *Then:*

$$t \models \phi \iff ok \in CT(Sat(C_\phi,t)).$$

PROOF. $\Rightarrow$) Induction on the structure of ϕ
a) ϕ is $\neg\psi$, $\psi \in \mathcal{L}_1$. We have $t \not\models \psi$. By lemma 8.6.6.1 $CT(Sat(C_\psi,t)) = \varnothing$. By rule 1:

$Sat(C_\phi,t) \xrightarrow{ok} Sat(C_\psi,t)$. Hence $ok \in CT(Sat(C_\phi,t))$.

b) ϕ is T. Rule 1 gives $Sat(C_T,t) \xrightarrow{ok} Sat(\delta,t) \not\rightarrow$. Hence $ok \in CT(Sat(C_\phi,t))$.

c) ϕ is $\phi_1 \wedge \phi_2$. Since $t \models \phi$ we also have $t \models \phi_1$ and $t \models \phi_2$. By induction $ok \in CT(Sat(C_{\phi_1},t))$ and $ok \in CT(Sat(C_{\phi_2},t))$. Since all outgoing transitions of $Sat(C_\phi,t)$ are proved using rule 2, one can easily see that $ok \in CT(Sat(C_\phi,t))$.

d) $\phi = \langle a \rangle \phi'$. Since $t \models \langle a \rangle \phi'$, there is a t' such that $t \xrightarrow{a} t'$ and $t' \models \phi'$. Induction gives that $ok \in CT(Sat(C_{\phi'},t'))$. Hence there is a termination node t'' such that $Sat(C_{\phi'},t') \xrightarrow{ok} t''$. Now an application of rule 3 gives that $ok \in CT(Sat(C_\phi,t))$.

$\Leftarrow$) Induction on the structure of ϕ.

a) ϕ is $\neg\psi$, $\psi \in \mathcal{L}_1$. If $Sat(C_\phi,t)$ does a move, then the last rule applied in the proof must have been rule 1 and the transition must be $Sat(C_\phi,t) \xrightarrow{ok} Sat(C_\psi,t)$. Because $ok \in CT(Sat(C_\phi,t))$, $Sat(C_\psi,t)$ can have no outgoing transitions. Since $\psi \in \mathcal{L}_1$, lemma 8.6.6.1 allows us to conclude that $t \not\models \psi$. Hence $t \models \phi$.

b) ϕ is T. Since $t \models T$ the implication holds.

c) ϕ is $\phi_1 \wedge \phi_2$. If $Sat(C_\phi,t)$ does a move then the last rule applied in the proof of this transition must have been rule 2. Since $ok \in CT(Sat(C_\phi,t))$, it must be that $ok \in CT(Sat(C_{\phi_1},t))$ and $ok \in CT(Sat(C_{\phi_2},t))$. But this means that we can apply the induction hypothesis to obtain $t \models \phi_1$ and $t \models \phi_2$. Hence $t \models \phi$.

d) ϕ is $\langle a \rangle \phi'$. If $Sat(C_\phi,t)$ does a move then the last rule applied in the proof must have been rule 3. So, because $ok \in CT(Sat(C_\phi,t))$, there are t',t'' with $t \xrightarrow{a} t'$, $Sat(C_{\phi'},t') \xrightarrow{ok} t''$ and t'' a termination node. This implies that $ok \in CT(Sat(C_{\phi'},t'))$. By induction $t' \models \phi'$. Hence $t \models \phi$. □

Now the second inclusion follows directly:

8.6.7. COROLLARY *(inclusion 2). Let P be a TSS in pure tyft/tyxt format. Then:*

$$\equiv_{\text{pure tyft/tyxt}} \subseteq \sim_{\mathcal{L}_2}.$$

8.7. In this section it will be shown that the inclusions 4,5 and 6 hold. As an immediate corollary it follows that inclusion 3 holds.

8.7.1. THEOREM *(inclusion 4). Let $\mathcal{A}=(S,A,\rightarrow)$ be a LTS. Then for all $s,t \in S$ and $m \in \mathbb{N}$:*

$$s \leftrightarrows^m t \quad \Rightarrow \quad s \sim_{\mathcal{L}_m} t.$$

PROOF. Suppose that $s \subsetneq^m t$ and $s \models \phi$ for some $\phi \in \mathcal{L}_m$. We prove that $t \models \phi$ with induction on m. The case $m=0$ is trivially ok. So suppose $m>0$. We prove $t \models \phi$ with induction on the structure of ϕ.

a) ϕ is $\neg\psi$, $\psi \in \mathcal{L}_{m-1}$. By definition of $s \subsetneq^m t$ we have $t \subsetneq^{m-1} s$. Application of the induction hypothesis gives $t \not\models \psi$ and hence $t \models \phi$.

b) ϕ is T. In this case $t \models \phi$ trivially holds.

c) ϕ is $\phi_1 \wedge \phi_2$. From $s \models \phi$ it follows that $s \models \phi_1$ and $s \models \phi_2$. By induction $t \models \phi_1$ and $t \models \phi$. Hence, $t \models \phi$.

d) ϕ is $\langle a \rangle \phi'$. There exists an s' such that $s \xrightarrow{a} s'$ and $s' \models \phi'$. Since $s \subsetneq^m t$, there exists an m-nested simulation R containing (s,t). Hence, for some $t' \in S$, $t \xrightarrow{a} t'$ and $s' R\, t'$. So $s' \subsetneq^m t'$. By induction $t' \models \phi'$ and thus $t \models \phi$. □

We define $\leftrightarrows^m$ and $\leftrightarrows^m_n$ as auxiliary notions. Roughly speaking, $s \leftrightarrows^m_n t$ means that s and t are m-nested simulation equivalent to depth n. $\leftrightarrows^m$ is the intersection of $\leftrightarrows^m_n$ for all n.

8.7.2. DEFINITION. Let $\mathcal{A}=(S,A,\rightarrow)$ be a LTS. Define relations $\subseteq_n^m \subseteq S\times S$ by:
- $s \subseteq_0^m t$ always,
- $s \subseteq_n^0 t$ always,
- $s \subseteq_{n+1}^{m+1} t$ iff $t \subseteq_{n+1}^m s$ and whenever $s \xrightarrow{a} s'$ then there is a t' such that $t \xrightarrow{a} t'$ and $s' \subseteq_n^{m+1} t'$.

We write $\leftrightsquigarrow_n^m t$ if $s \subseteq_n^m t$ and $t \subseteq_n^m s$, $s \leftrightsquigarrow^m t$ if for all n: $s \leftrightsquigarrow_n^m t$, and $s \subseteq^m t$ if for all n: $s \subseteq_n^m t$.

8.7.3. LEMMA. *For all $m,n\geqslant 0$: $\subseteq_{n+1}^m \subseteq \subseteq_n^m$ and $\leftrightsquigarrow_{n+1}^m \subseteq \leftrightsquigarrow_n^m$.*
PROOF. Straightforward simultaneous induction on m and n. □

8.7.4. THEOREM *(inclusion 5). Let $\mathcal{A}=(S,A,\rightarrow)$ be a LTS which is image finite. Then for all $s,t\in S$ and $m\in\mathbb{N}$:*

$$s \sim_{\mathcal{L}_m} t \quad \Rightarrow \quad s \leftrightsquigarrow^m t.$$

PROOF. Suppose that $s \not\subseteq^m t$. We show that there is a $\phi\in\mathcal{L}_m$ such that $s\models\phi$ but $t\not\models\phi$ by induction on m. It cannot be that $m=0$. So take $m>0$. Since $s \not\subseteq^m t$, there must be an n such that $s \not\subseteq_n^m t$. With induction on n we show that there exists a ϕ such that $s\models\phi$ but not $t\models\phi$.
It cannot be that $n=0$. Take $n>0$. One reason why $s \not\subseteq^m t$ is that $t \not\subseteq_n^{m-1} s$. If this is the case then we can find by induction a $\psi\in\mathcal{L}_{m-1}$ such that $t\models\psi$ and $s\not\models\psi$. Hence $s\models\neg\psi$ (the formula $\neg\psi$ is in $\mathcal{L}_m$) and $t\not\models\neg\psi$. If $t \not\subseteq_n^{m-1} s$, then it must be that for some $a\in A$ and $s'\in S$ with $s \xrightarrow{a} s'$ we have that for all t' with $t \xrightarrow{a} t'$: $s' \not\subseteq_{n-1}^{m-1} t'$. Now a first possibility is that there is no t' such that $t \xrightarrow{a} t'$. In this situation $s\models\langle a\rangle T$, $t\not\models\langle a\rangle T$ and we are done. The other possibility is that there is a nonzero, but due to the image finiteness, finite number of states $t_1,..,t_p$ that can be reached from t by an a-transition. Since $s' \not\subseteq_{n-1}^{m-1} t_i$ for $1\leqslant i\leqslant p$, we have by induction that there are $\phi_i\in\mathcal{L}_m$ such that $s'\models\phi_i$ and $t_i\not\models\phi_i$. Consider the $\mathcal{L}_m$-formula $\phi=\phi_1\wedge..\wedge\phi_p$. Since $s'\models\phi$ and $t_i\not\models\phi$, $s\models\langle a\rangle\phi$ and $t\not\models\langle a\rangle\phi$. □

8.7.5. THEOREM *(inclusion 6). Let $\mathcal{A}=(S,A,\rightarrow)$ be a LTS which is image finite. Then for all $s,t\in S$ and $m\in\mathbb{N}$:*

$$s \leftrightsquigarrow^m t \quad \Rightarrow \quad s \leftrightarrows^m t$$

PROOF. Suppose that $s\subseteq^m t$. With induction on m we prove that $s \subseteq^m t$. The case $m=0$ is trivial. So suppose $m>0$. We prove that $\subseteq^m$ is an m-nested simulation relation. Whenever $v \subseteq^m w$ then for all n, $v \subseteq_n^m w$. Hence by definition of $\subseteq^m$, $w \subseteq_n^{m-1} v$ for all n. Thus $w \subseteq^{m-1} v$ and by induction $w \subseteq^{m-1} v$. So the relation $\subseteq^m$ is contained in the relation $(\subseteq^{m-1})^{-1}$. It remains to be shown that $\subseteq^m$ is a simulation relation. Suppose $v \subseteq^m w$ and $v \xrightarrow{a} v'$. Since for all $n>0$, $v \subseteq_n^m w$ there is for each n a w_n such that $w \xrightarrow{a} w_n$ and $v' \subseteq_{n-1}^m w_n$. Due to the image finiteness there must be a w^* that occurs infinitely often in the sequence $w_1, w_2,....$ Because for all n $\subseteq_{n-1}^m \supseteq \subseteq_n^m$ by lemma 8.7.3, we have that for all $n>0$, $v \subseteq_{n-1}^m w^*$ and therefore $v \subseteq^m w^*$. This concludes the proof that $\subseteq^m$ is an m-nested simulation. □

8.7.6. COROLLARY. *Let $\mathcal{A}=(S,A,\rightarrow)$ be a LTS which is image finite. Then for all $s,t\in S$ and $m\in\mathbb{N}$:*

$$s \leftrightarrows^m t \quad \Leftrightarrow \quad s \leftrightsquigarrow^m t \quad \Leftrightarrow \quad s \sim_{\mathcal{L}_m} t.$$

8.8. Trace congruence. Using the above results, we can easily characterize the 'trace congruence' induced by pure *tyft/tyxt* rules as simulation equivalence or $\mathcal{L}_1$ formula equivalence (for image finite LTS's). We just do the argumentation above over again for trace congruence instead of completed trace congruence. First the notion of trace congruence is defined.

8.8.1. DEFINITION. Let $\mathcal{L}=(S,A,\rightarrow)$ be a LTS. A sequence $\sigma\in A^*$ is a *trace* of s if there are actions $a_1,a_2,..,a_n\in A$ with $\sigma=a_1{*}a_2{*}..{*}a_n$ and states $s_1,s_2,..,s_n\in S$ such that $s\xrightarrow{a_1}s_1\xrightarrow{a_2}..\xrightarrow{a_n}s_n$. $T(s)$ is the set of all traces of s. Two states $s,t\in S$ are *trace equivalent* if $T(s)=T(t)$. This is denoted as $s\equiv_T t$.

8.8.2. DEFINITION. Let $\mathcal{F}$ be some format of TSS rules. Let $P=(\Sigma,A,R)$ be a TSS in $\mathcal{F}$ format. Two terms $s,t\in T(\Sigma)$ are *trace congruent with respect to* $\mathcal{F}$ rules if for every TSS $P'=(\Sigma',A',R')$ in $\mathcal{F}$ format which can be added conservatively to P and for every $\Sigma\oplus\Sigma'$-context $C[\,]$: $C[s]\equiv_T C[t]$.

In fact most of the work is already done. This can easily be seen if we look at figure 11, but instead of 2-nested simulation equivalence we now have (1-nested) simulation equivalence and instead of $\mathcal{L}_2$-equivalence $\mathcal{L}_1$- equivalence. Inclusion 3 is already proved as the right triangle of figure 11 is proved for all m. Inclusion 1 follows by lemma 8.5.6 and the observation that simulation equivalence refines trace equivalence. Inclusion 2 can be proved using the same test system as in the proof of theorem 8.6.6.

8.9. Characterization theorem for tree rules. The characterization theorem 8.4.4 for tree rules of Bloom, Istrail & Meyer follows from theorem 8.5.7, corollary 8.7.6 and the following theorem 8.9.1. In fact this combination gives a result which is even stronger than the result of Bloom, Istrail & Meyer as we allow more general rules in the original system and our test system is finite if the alphabet of the old system is finite (they did not look at finite test systems for $\mathcal{L}_2$ formulas). The next theorem also strengthens theorem 8.6.6 because now only tree rules are used. But, as the proof of this theorem is rather tricky, we liked to give the simpler variant first.

8.9.1. THEOREM. *Let $P_0=(\Sigma_0,A_0,R_0)$ be a TSS in pure tyft/tyxt format. Then there is a TSS $P_1=(\Sigma_1,A_1,R_1)$ in tree rule format, which can be added conservatively to P_0, such that completed trace congruence within $P_0\oplus P_1$ is included in $\mathcal{L}_2$ formula equivalence. Moreover, if alphabet A_0 is finite, then the components of P_1 are finite too.*

PROOF (sketch). The alphabet A_1 consists of A_0 together with 8 new labels:

$$A_1=A_0\cup\{ok,ko,left,right,size,neg,\langle\rangle,i\}.$$

Σ_1 contains δ, $+$ and prefix-operators $a:$ for every $a\in A_1$. In R_1 we find the usual rules for these operators. Furthermore Σ_1 contains binary operators $\|_H$ which model parallel composition with synchronisation of actions in a set $H\subseteq A_1$. For these operators R_1 contains rules ($a\in A_1$):

$$\frac{x\xrightarrow{a}x'}{x\|_H y\xrightarrow{a}x'\|_H y}\quad a\notin H\qquad \frac{y\xrightarrow{a}y'}{x\|_H y\xrightarrow{a}x\|_H y'}\quad a\notin H$$

$$\frac{x\xrightarrow{a}x',\ y\xrightarrow{a}y'}{x\|_H y\xrightarrow{a}x'\|_H y'}\qquad a\in H$$

Next Σ_1 contains a binary operator *Sat* which tests whether its second argument satisfies the $\mathcal{L}_2$ formula which is encoded using the rules below. Further it contains the auxiliary operators *Context*, *skip-i* and *ok-to-ko*. The rules in R_1 for these operators are displayed in table 5 (where $a\in A_1$). Clearly, if A_0 is finite then A_1, Σ_1 and R_1 are finite too. Let the mapping $s:\mathcal{L}_1\rightarrow\mathbb{N}$ be given by:

$$s(T)=0$$
$$s(\phi\wedge\psi)=1+s(\phi)+s(\psi)$$
$$s(\langle a\rangle\phi)=1+s(\phi)$$

and let the Σ_1 terms S_n ($n\geqslant 0$) be given by:

$\dfrac{x \xrightarrow{ok} x'}{Sat(x,y) \xrightarrow{ok} \delta}$	$\dfrac{x \xrightarrow{i} x'}{Context(x,y) \xrightarrow{i} Context(x', skip\text{-}i(y))}$
$\dfrac{x \xrightarrow{left} x_l,\ x \xrightarrow{right} x_r}{Sat(x,y) \xrightarrow{i} Sat(x_l,y) \|_{\{ok\}} Sat(x_r,y)}$	$\dfrac{x \xrightarrow{ok} x'}{Context(x,y) \xrightarrow{ok} ok\text{-}to\text{-}ko(y)}$
$\dfrac{x \xrightarrow{\langle\rangle} x' \xrightarrow{a} x'',\ y \xrightarrow{a} y'}{Sat(x,y) \xrightarrow{i} Sat(x'',y')}$	$\dfrac{x \xrightarrow{i} x' \xrightarrow{a} x''}{skip\text{-}i(x) \xrightarrow{a} x''}$
$\dfrac{x \xrightarrow{size} x',\ x \xrightarrow{neg} x''}{Sat(x,y) \xrightarrow{i} Context(x', Sat(x'',y))}$	$\dfrac{x \xrightarrow{ok} x'}{ok\text{-}to\text{-}ko(x) \xrightarrow{ko} \delta}$

TABLE 5

$$S_0 = ok{:}\delta$$

$$S_{n+1} = i{:}S_n$$

$\mathcal{L}_2$ formulas are coded as follows:

$$C_T = ok{:}\delta$$

$$C_{\phi\wedge\psi} = left{:}C_\phi + right{:}C_\psi$$

$$C_{\neg\phi} = size{:}S_{s(\phi)} + neg{:}C_\phi$$

$$C_{\langle a\rangle\phi} = \langle\rangle{:}a{:}C_\phi$$

We will now briefly explain the way in which the above construction works. We have the following claim:

8.9.1.1. CLAIM. *Let $\phi\in\mathcal{L}_2$ and $t\in T(\Sigma_0\oplus\Sigma_1)$. Then $t\models\phi$ iff $Sat(C_\phi,t)$ has a completed trace with an ok action but without a ko action.* □

It is not hard to see that the above claim is correct in case $\phi\in\mathcal{L}_1$. This is a direct consequence of the next claim which can be proved easily by means of induction on the structure of ϕ:

8.9.1.2. CLAIM. *Let $\phi\in\mathcal{L}_1$ with $s(\phi)=n$ and let $t\in T(\Sigma_0\oplus\Sigma_1)$. Then:*
- $t\models\phi \Rightarrow \{i^n{\star}ok\}\subseteq CT(Sat(C_\phi,t))\subseteq\{i^n{\star}ok\}\cup\{i^m \mid 1\leq m\leq n\}$,
- $t\not\models\phi \Rightarrow CT(Sat(C_\phi,t))\subseteq\{i^m \mid 1\leq m\leq n\}$.

The problem is what to do with negations. The key idea of our solution is that if one applies the *skip-i* operator $s(\phi)$ times on $Sat(C_\phi,t)$, the trace set of the resulting process consists of *ok* if $t\models\phi$ and will be empty otherwise. So what we have to do is to place $s(\phi)$ times a *skip-i* operator around $Sat(C_\phi,t)$ in a structured way and next apply a renaming of *ok* into *ko*. This is of course done using the binary operator *Context*. The first argument of this operator gives instruction to build a context around the second argument. In case a formula $\neg\phi$ has to be tested, our construction works in such a way that (after some *i*-steps) always an *ok* step will be generated, whereas a subsequent *ko* action is generated only when the tested process satisfies ϕ. Hence we claim that:

8.9.1.3. CLAIM. *Let* $\phi \in \mathbb{L}_1$ *with* $s(\phi)=n$ *and let* $t \in T(\Sigma_0 \oplus \Sigma_1)$. *Then:*
- $t \models \phi \Rightarrow CT(Context(S_n, Sat(C_\phi, t))) = \{i^n \star ok \star ko\}$,
- $t \not\models \phi \Rightarrow CT(Context(S_n, Sat(C_\phi, t))) = \{i^n \star ok\}$.

Using claim 8.9.1.3, claim 8.9.1.1 can be proved with straightforward induction on the structure of ϕ.

9. COMPARISON WITH OTHER FORMATS

In this section we will give an extensive comparison of our format with the formats proposed by DE SIMONE [29,30] and BLOOM, ISTRAIL & MEYER [10]. First both formats will be described.

9.1. DEFINITION (cf. [29]). Let $\Sigma=(F,r)$ be a signature and let A be a set of labels. A *dS rule* (over Σ and A) takes the form:

$$\frac{\{x_i \xrightarrow{a_i} y_i \mid i \in I\}}{f(x_1,..,x_n) \xrightarrow{a} t}$$

where:
- $f \in F$ and $r(f)=n$,
- $I \subseteq \{1,..,n\}$,
- $x_1,..,x_n, y_i$ $(i \in I)$ are distinct variables,

Let for $1 \leqslant i \leqslant n$ $x_i'=y_i$ if $i \in I$ and $x_i'=x_i$ otherwise.
- t is a term in $T(\Sigma, \{x_1',..,x_n'\})$ in which each x_i' occurs at most once.

Clearly the dS format as presented above is included in our *tyft/tyxt* format. One should note however that De Simone assumes in addition that the set of labels is an (infinite) commutative monoid. Moreover he includes (unguarded) recursion in the language together with the standard fixed point rules.

9.2. DEFINITION (cf. [10]). Let $\Sigma=(F,r)$ be a signature and let A be a set of labels. A *GSOS rule* (over Σ and A) takes the form:

$$\frac{\{x_i \xrightarrow{a_{ij}} y_{ij} \mid 1 \leqslant i \leqslant l,\ 1 \leqslant j \leqslant m_i\} \cup \{x_i \xrightarrow{b_{ij}} \mid 1 \leqslant i \leqslant l,\ 1 \leqslant j \leqslant n_i\}}{f(x_1,..,x_l) \xrightarrow{a} t}$$

where the variables are all distinct, $f \in F$, $l=r(f)$, $m_i, n_i \geqslant 0$, $a_{ij}, b_{ij} \in A$ and t is a term in $T(\Sigma, \{x_i, y_{ij} \mid 1 \leqslant i \leqslant l,\ 1 \leqslant j \leqslant m_i\})$.

A *GSOS rule system* is a 3-tuple (Σ, A, R) with Σ a signature, A a set of labels and R a set of GSOS rules over Σ and A.

We should mention here that the above definition is simplified in order to make comparison possible and only gives an approximation of the notion of a GSOS rule system as it is defined in [10]. There a GSOS rule system contains some additional ingredients for dealing with guarded recursion and there are a number of finiteness constraints. The feature which distinguishes GSOS rules from the other rules in this paper is the possibility of *negative* premises. This makes that it is not immediately clear how (and if) a GSOS rule system determines a transition relation.

9.2.1. DEFINITION. Let (Σ, A, R) be a GSOS rule system. A transition relation $\longrightarrow \subseteq T(\Sigma) \times A \times T(\Sigma)$ *agrees with* the rules in R if:
- Whenever an instantiation by a substitution σ of the premises of a rule is true of the relation, then the instantiation of the conclusion by σ is true as well.
- Whenever $t \xrightarrow{a} t'$ is true, then there is a rule r and an instantiation σ such that $t \xrightarrow{a} t'$ is the instantiation of the conclusion of r by σ, and the instantiations of the premises of r by σ are true.

It is not hard to show that for any GSOS rule system, there is a unique transition relation which agrees with the rules. If a GSOS rule system only contains positive rules then it is a TSS according to our definition. Moreover in this case the unique transition relation which agrees with the rules according to the definition above is just the same relation as the one defined in definition 3.2 using the notion of proof trees of transitions.
The following example from [10] shows that the GSOS format cannot be combined consistently with the *tyft/tyxt* format. There are 4 operators in the signature: f, g, c and d. We have an action a and the following rules:

$$\frac{x \xrightarrow{a} y \quad y \xrightarrow{a} z}{f(x) \xrightarrow{a} d}$$

$$\frac{x \not\xrightarrow{a}}{g(x) \xrightarrow{a} d}$$

$$c \xrightarrow{a} g(f(c))$$

There is no transition relation which agrees with these rules. In particular, $f(c)$ can move iff it cannot move.

9.3. EXAMPLES. Below we list some applications that illustrate the differences between the formats.

9.3.1. Global closure properties. Rules in *tyxt* format neither fit into the dS format nor in the GSOS format. One could say that *tyxt* rules, like for instance the τ rules of table 2, express certain 'global closure properties', a form of operational behaviour which is in general independent of the particular function symbol at the head of a term. An operator that can be defined using closure properties is the 'atomic version' operator that was introduced by DE BAKKER & KOK [6] for giving semantics to concurrent Prolog. We cannot exactly model this operator as we would then need an infinite number of premises in our rules. Therefore, we will give here our own variant using our own notation. Take as starting point the signature of BPA$_\delta$. But as labels of transitions we now don't take elements of $Act_\surd$, but elements of the set of finite sequences over $Act_\surd$. We write a for the sequence consisting of the single symbol $a \in Act_\surd$. With $\sigma\sigma'$ we denote the concatenation of the sequences σ and σ'. The set of rules of the TSS contains the rules of R(BPA$_\delta$) (but now the labels should be interpreted as sequences!) and moreover the following rules:

$$\frac{x \xrightarrow{\sigma} y \quad y \xrightarrow{\sigma'} z}{x \xrightarrow{\sigma\sigma'} z}$$

$$\frac{x \xrightarrow{\sigma\surd} y}{[x] \xrightarrow{\sigma\surd} y}$$

The first rule expresses that sequences from $Act_\surd$ are allowed. The last rule expresses that only successful sequences, this are the sequences ending on $\surd$, can happen in the scope of an atomic version operator. These rules are in *tyft/tyxt* format. Hence, strong bisimulation is a congruence in this setting.

9.3.2. Contexts. Often it is very useful to have function symbols in the left hand side of a premise. However, this is not allowed by the dS and GSOS format. In section 6.2 we saw that these rules can be used to model recursion. Also in the system of table 4 for testing $\mathbb{L}_2$ formulas, this type of rules play an important role. In BAETEN & VAN GLABBEEK [5], operators ϵ_K are described that erase all actions from a set $K \subseteq Act$. We can add these operators to P(BPA$_\delta$) together with the following rules from [5]:

$$\frac{x \xrightarrow{a} y}{\epsilon_K(x) \xrightarrow{a} \epsilon_K(y)} \quad a \notin K$$

$$\frac{x \xrightarrow{a} y \quad \epsilon_K(y) \xrightarrow{b} z}{\epsilon_K(x) \xrightarrow{b} z} \quad a \in K$$

The same type of trick can also be used to describe the atomic version operator of the previous section without *tyxt* rules:

$$\frac{x \xrightarrow{\surd} y}{[x] \xrightarrow{\surd} y}$$

$$\frac{x \xrightarrow{a} y \quad [y] \xrightarrow{\sigma} z}{[x] \xrightarrow{a\sigma} z}$$

9.3.3. Lookahead. All operators defined with the dS or GSOS format have a lookahead of at most 1. Hence the following operator, which can be viewed as the inverse of the split operator of [15], cannot be defined:

$$\frac{x \xrightarrow{a^+} y \xrightarrow{a^-} z}{combine(x) \xrightarrow{a} combine(z)}$$

Other examples of operators with a large lookahead are the ϵ_K and the atomic version operator as described above. As a last example we mention the *abstraction* or *hiding* operator from ACP_τ [13] (here $I \subseteq Act$):

$$\frac{x \xrightarrow{a} x'}{\tau_I(x) \xrightarrow{\tau} \tau_I(x')} \quad a \in I$$

$$\frac{x \xrightarrow{a} x'}{\tau_I(x) \xrightarrow{a} \tau_I(x')} \quad a \notin I$$

If we add these rules to the system $P(BPA^\tau_{\epsilon\delta})$ as described in section 6.1, then we can derive:

$$\tau_{\{i\}}(i \cdot a) \xrightarrow{a} \tau_{\{i\}}(\epsilon).$$

9.3.4. Non-linearity. In contrast to the dS format, the GSOS format and also our format can describe operators which copy their arguments. Below we present an example where such an operator occurs naturally. It describes an operational semantics of the natural numbers which is based on the idea of counting: the process associated to an integer expression performs as many actions as the value which is denoted by this expression under the standard interpretation. We consider the signature containing a constant name 0, a unary function name *succ* and binary operators $+$ and $\times$. There is only one transition label, namely 1. The operational semantics of the operators is described by the following rules:

$$succ(x) \xrightarrow{1} x$$

$$\frac{x \xrightarrow{1} x'}{x+y \xrightarrow{1} x'+y} \qquad \frac{y \xrightarrow{1} y'}{x+y \xrightarrow{1} x+y'}$$

$$\frac{x \xrightarrow{1} x' \quad y \xrightarrow{1} y'}{x \times y \xrightarrow{1} (x' \times y') + (x' + y')}$$

Observe that two expressions denote the same value under the standard interpretation iff they are bisimilar.

9.3.5. Catalysis. We can add to $P(\text{BPA}_\delta)$ the following rule which fits in the GSOS format and in the *tyft* format:

$$\frac{x \xrightarrow{ok} x',\ y \xrightarrow{a} y'}{Cat(x,y) \xrightarrow{a} Cat(x,y')}$$

Here we have a situation, not allowed by De Simone, where a potential *ok*-action of the first component makes it possible for the second component to proceed. But when it proceeds the first component remains unchanged. Hence, one can view the first component as a 'catalyst' of the second component.

9.3.6. Priorities. In BAETEN, BERGSTRA & KLOP [4] an operator is introduced to describe priorities in ACP, whereby some actions have priorities over others in a non-deterministic choice. The operator turns out to be quite interesting and has been used in a number of applications. In [4] the operator is defined using equations, but if one uses Plotkin-style rules then it seems inevitable (and also very natural) to use negative hypotheses.

Consider the GSOS rule system $P(\text{BPA}_\delta)$ and assume that the set $Act_\surd$ of labels is finite. Assume furthermore that a partial order $>$ is given on $Act_\surd$ such that $\surd$ is not in the ordering. Now we can add a unary operator θ to the rule system, with for each $a \in A_\surd$ a rule:

$$\frac{x \xrightarrow{a} x',\ \forall b > a : x \not\xrightarrow{b}}{\theta(x) \xrightarrow{a} \theta(x')}$$

The rule expresses that in the scope of a θ-operator an a action can occur unless an action with a higher priority is possible. Another example of an operator that is defined using rules with negative premises is the *broadcast* operator as described by PNUELI [28].

9.4. Completed trace congruences. The differences between the formats presented thus far can be understood also if we look at the completed trace congruences which they induce. In section 8 we saw that the trace congruence induced by (variants) of the pure *tyft/tyxt* format coincides with $\pounds_2$ formula equivalence.

The main theorem which De Simone proved about his format is that all operators defined using his type of inductive rules can also be defined by MEIJE-SCCS 'architectural' expressions. Similar results have not yet been proved for the GSOS and the *tyft/tyxt* format. Now it is a standard result that the completed trace congruence induced by languages like MEIJE-SCCS, ACP, CSP, etc., coincides with *failure equivalence* ($\equiv_F$) (see for instance [8]). Hence the completed trace congruence induced by the dS format is failure equivalence.

BLOOM, ISTRAIL & MEYER [10] characterized the completed trace congruence induced by their format in terms of the equivalence corresponding to the following set of formulas[1]:

9.4.1. DEFINITION. The set $\mathfrak{D}$ of *denial (HML) formulas* (over a given alphabet $A = \{a,b,\ldots\}$) is given by the following grammar:

$$\phi ::= T \mid \phi \wedge \phi \mid [a]F \mid \langle a \rangle \phi.$$

1. The formulas as defined in [10] were called *limited modal formulas* and may also contain F and $\vee$. However, it is easily proved that this addition does not increase the distinguishing power.

9.4.2. THEOREM (BLOOM, ISTRAIL & MEYER). *Let $P=(\Sigma,A,R)$ be a GSOS rule system such that the associated transition system is image finite. Then:* $\equiv_{GSOS} = \sim_{\mathfrak{D}}$.

Some additional insight is provided by the following characterization of denial equivalence which is due to LARSEN & SKOU [20].

9.4.3. DEFINITION. Let $\mathcal{C}=(S,A,\rightarrow)$ be a LTS. A relation $R\subseteq S\times S$ is *2/3-bisimulation* if it satisfies:
1. whenever $s\,R\,t$ and $s\xrightarrow{a}s'$ then, for some $t'\in S$, also $t\xrightarrow{a}t'$ and $s'R\,t'$,
2. whenever $s\,R\,t$ and $t\xrightarrow{a}t'$ then, for some $s'\in S$, also $s\xrightarrow{a}s'$.

Two states $s,t\in S$ are *2/3-bisimilar* in $\mathcal{C}$ if there exists a 2/3-bisimulation containing the pair (s,t) and a 2/3-bisimulation containing the pair (t,s).

9.4.4. THEOREM (LARSEN & SKOU). *Let $\mathcal{C}=(S,A,\rightarrow)$ be a LTS. Then two states are 2/3-bisimilar just in case they satisfy exactly the same denial formulas.*

It is a trivial exercise to show that:

$$\leftrightarrows^2 \subseteq \leftrightarrows_{2/3} \subseteq \equiv_F \subseteq \equiv_{CT}$$

The following examples show that these inclusions are strict:

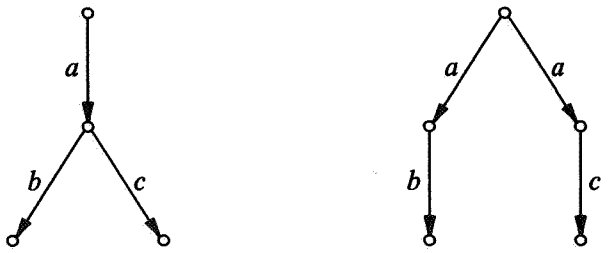


FIGURE 13. Completed trace equivalent but not dS congruent

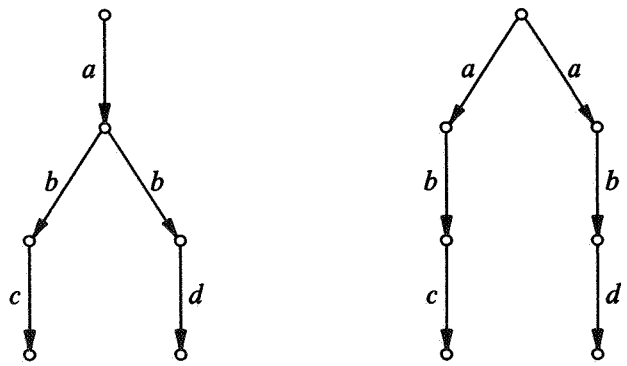


FIGURE 14. dS congruent but not GSOS congruent

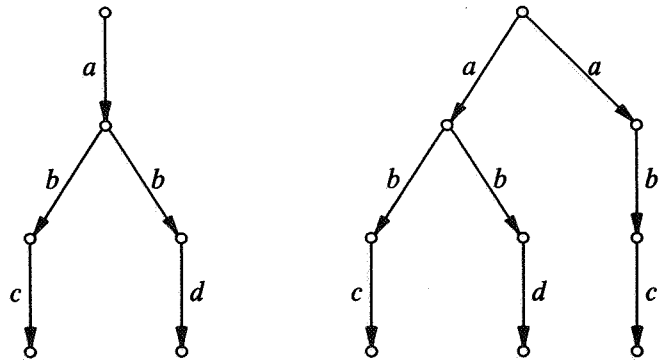


FIGURE 15. GSOS congruent but not pure *tyft/tyxt* congruent

9.4.5. Testing denial formulas. The question arises whether all features of the GSOS format are really needed in order to test denial formulas. In particular it is interesting to know whether the negative premises add anything to the discriminating power of the format. Surprisingly, as was first observed by ROB VAN GLABBEEK [14], this is not the case: GSOS congruence coincides with positive GSOS congruence. Below we present a system in positive GSOS format for testing denial formulas. The system is simpler than the original system of Rob van Glabbeek. Moreover our system has the advantage of being finite in case the alphabet of the old system is finite.

9.4.6. THEOREM. *Suppose we have a TSS $P_0=(\Sigma_0,A_0,R_0)$ in GSOS format. Then there exists a TSS $P_1=(\Sigma_1,A_1,R_1)$ in GSOS format with all premises positive and non-branching, which can be added conservatively to P_0, such that completed trace congruence within $P_0 \oplus P_1$ is included in denial equivalence. Moreover, if alphabet A_0 is finite, then the components of P_1 are finite too.*

PROOF. The set A_1 consists of A_0 together with 6 new labels:

$$A_1 = A_0 \cup \{ok, ko, left, right, [], \langle\rangle\}.$$

Signature Σ_1 contains a constant δ, unary function names $a:$ for each $a \in A_1$, and binary function names $+$, $\|$, Sat, $Sat_{[]}$, $Sat_{\langle\rangle}$ and Sat_{right}. The δ, $a:$, $+$ are as usual. $\|$ is just arbitrary interleaving. The Sat operator tests whether its second argument satisfies the denial formula which is represented by its first argument. The rules for the $\|$-operator and the various Sat-operators are given in table 6. In the table a ranges over A_1. Check that P_1 can be added conservatively to P_0.

$$\frac{x \xrightarrow{[]} x'}{Sat(x,y) \xrightarrow{[]} Sat_{[]}(x',y)} \qquad \frac{x \xrightarrow{a} x' \quad y \xrightarrow{a} y'}{Sat_{[]}(x,y) \xrightarrow{ko} \delta}$$

$$\frac{x \xrightarrow{\langle\rangle} x'}{Sat(x,y) \xrightarrow{\langle\rangle} Sat_{\langle\rangle}(x',y)} \qquad \frac{x \xrightarrow{a} x' \quad y \xrightarrow{a} y'}{Sat_{\langle\rangle}(x,y) \xrightarrow{ok} Sat(x',y')}$$

$$\frac{x \xrightarrow{left} x'}{Sat(x,y) \xrightarrow{left} Sat(x',y) \| Sat_{right}(x,y)} \qquad \frac{x \xrightarrow{right} x'}{Sat_{right}(x,y) \xrightarrow{right} Sat(x',y)}$$

$$\frac{x \xrightarrow{a} x'}{x \| y \xrightarrow{a} x' \| y} \qquad \frac{y \xrightarrow{a} y'}{x \| y \xrightarrow{a} x \| y'}$$

TABLE 6

Denial formulas are encoded using the following rules:

$$C_T = \delta$$
$$C_{\phi \wedge \psi} = left{:}C_\phi + right{:}C_\psi$$
$$C_{[a]F} = []{:}a{:}\delta$$
$$C_{\langle a \rangle \phi} = \langle\rangle{:}a{:}C_\phi$$

CLAIM. *Let $t \in T(\Sigma_0 \oplus \Sigma_1)$ and let ϕ be a denial formula. Then $t \models \phi$ iff $Sat(C_\phi, t)$ has a completed trace with as many ok's as ϕ has $\langle a \rangle$'s, and no ko.*

PROOF. Rather straightforward induction on the structure of ϕ. □

9.4.7. Comparison of testing abilities. The notion of testing which underlies CCS/CSP/ACP, and hence De Simone's format, is well-known (see for instance [8, 12]): these languages allow one to observe *traces* and to detect *refusals* indirectly: one concludes that a certain action is refused because some completed trace is not there. The construction in the proof of theorem 9.4.6 clearly shows which notion of testing underlies the (positive) GSOS format: the format allows one to observe *traces* of processes, to detect *refusals* and to make *copies* of processes at every moment. In the general GSOS format refusals can be observed directly: one can define a context which performs an *ok* step if its argument cannot do a certain action. In the positive GSOS format refusals can also be observed, but only indirectly. The key feature which distinguishes the positive GSOS format from De Simone's format is the capacity to make copies of processes at every moment. Observe that the only rule in table 6 that does not fit in De Simone's format is the rule dealing with the *left* action. In this rule the x and y are copied. In many situations copying is a natural operation which can be realised physically by for instance a core dump procedure.

The construction in the proof of theorem 8.9.1 shows that the additional testing power needed to bring one from denial equivalence to $\mathfrak{L}_2$ formula equivalence only consists of the ability to see whether some action is possible in the future: there should be operations with a *lookahead* (in fact the proof of theorem 8.9.1 shows that a lookahead of 2 is already enough). Using operators with a lookahead one can investigate *all* branches of a process for positive information and one can see whether a certain *tree* is possible. In particular one can see whether there exists a branch in which a certain action is present. In the same way as one can observe in the dS format that a certain action is refused because some completed trace is not there, one can conclude in the *tyft/tyxt* format from the absence of some completed trace that a tree is refused. The ability to see in the future of a process can be considered as a weak form of *global testing*. Global testing is the same as what MILNER [22] calls *controlling the weather conditions*. ABRAMSKY [1] describes global testing as: "the ability to enumerate all (of finitely many) possible 'operating environments' at each stage of the test, so as to guarantee that all nondeterministic branches will be pursued by various copies of the subject process". Because an operator with lookahead is not capable to see negative information (like the absence of some action) directly, and because it is also not able to force that all nondeterministic branches are pursued by some number of copies, lookahead does not give one the full testing power of global testing. Since global testing is needed in order to distinguish between processes which are not bisimilar, this explains why the fully abstract semantics induced by our format is still below bisimulation equivalence. Global testing in the above sense seems very unrealistic as a testing ability and in direct conflict with the observational viewpoint of concurrent systems. Recently however, LARSEN & SKOU [20] have pointed out that if one assumes that every transition in a transition system has a certain minimum probability of being taken, an observer can - due to the probabilistic nature of transitions - with arbitrary high degree of confidence, assume that all transitions have been examined, simply by repeating an experiment many times (using the copying facility). This idea gives some plausibility to the notion of global testing. In fact Larsen & Skou deviced some testing algorithms which allow them, with a probability arbitrary close to 1, to distinguish between processes that are not bisimilar.

Unless one believes in fortune telling as a technique which has some practical relevance for computer science, lookahead as a testing notion is not very realistic. Still, this lookahead pops up naturally if one looks at the maximal format of rules for which bisimulation is a congruence and we showed that rules with a lookahead are often useful. Therefore we think that, just like bisimulation equivalence, $\mathcal{L}_2$ formula equivalence is an interesting equivalence that is worth studying, even though it does not correspond to a very natural notion of testing.

9.4.8. Finiteness and decidability. In their paper 'Bisimulation can't be traced', BLOOM, ISTRAIL & MEYER [10] argue that bisimulation *cannot* be reduced to completed trace congruence with respect to any *reasonably structured* system of process constructing operations. They present the GSOS format, which they believe to be the most general format leading to reasonably structured systems, and then show that the congruence induced by this format is denial formula equivalence. Although the pure *tyft/tyxt* format cannot trace bisimulation equivalence, it can trace more of it than the GSOS format. This implies that not all pure *tyft/tyxt* rules are structured according to the definition of Bloom, Istrail & Meyer. And indeed what's wrong in their opinion with our rules is that they might lead to transition systems with a transition relation which is infinitely branching or not computable. The various finiteness constraints which are present in the definition of the GSOS format in [10], are motivated by the requirement that the transition relation should be computably finitely branching. We think that, although it is certainly pleasant to have finiteness and decidability, it is much too strong to call any TSS leading to a transition relation which does not have these properties 'not reasonably structured' (this is what Bloom, Meyer & Istrail seem to do in their paper). Since our format gives us the expressiveness to describe the invisible nature of τ (see section 6.1) it is to be expected that, in general, we also have the infinite branching and undecidability of the models of CCS/ACP$_\tau$ based on observational congruence. If one disqualifies infinitary and undecidable TSS's right from the start, then one misses a large number of interesting applications. Of course the question what type of TSS's do lead to computably finitely branching transition systems is a very interesting one. It seems that if one generalises the positive GSOS format in the direction of the *tyft/tyxt* format, infinite branching arises quite soon. The following example for instance, which is due to Bard Bloom, illustrates that function symbols in the premises are 'dangerous'.

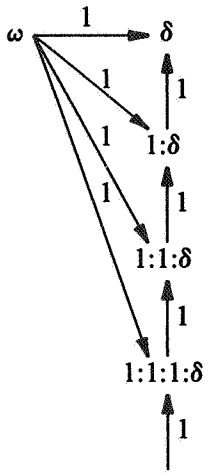


FIGURE 15

In the example we have prefixing and δ as usual and moreover a constant ω with rules:

$$\omega \xrightarrow{1} \delta \qquad \frac{\omega \xrightarrow{1} x}{\omega \xrightarrow{1} 1{:}x}$$

The part of the transition system which is displayed in figure 15 shows that ω has an infinite number of outgoing transitions. Another example illustrating the same point is obtained by adding recursion to $P(BPA_\delta^\S)$ in the style of section 6.2 with the 'unguarded' recursive definition $X \Leftarrow Xa + a$. It is easy to give examples of *tyxt* rules or tree rules which lead to infinite branching or undecidability. It is an open question to find a format in between positive GSOS and *tyft/tyxt* which always leads to computably finitely branching transition relations.
In our view one reason why rules with a lookahead are important is that they make it possible to have different levels of granularity of actions and to express that an action at one level can be composed of several smaller actions at a lower level. The system of table 6 for testing denial equivalence is an excellent example of a situation where the GSOS format forces one to do in two steps what one would like to do in only one.

REFERENCES

[1] S. ABRAMSKY (1987): *Observation equivalence as a testing equivalence.* Theoretical Computer Science 53, pp. 225-241.

[2] P. AMERICA, J.W. DE BAKKER, J.N. KOK & J.J.M.M. RUTTEN (1986): *Operational semantics of a parallel object-oriented language.* In: Conference Record of the 13th ACM Symposium on Principles of Programming Languages, St. Petersburg, Florida, pp. 194-208.

[3] J.C.M. BAETEN & J.A. BERGSTRA (1988): *Global Renaming Operators in Concrete Process Algebra.* I&C 78(3), pp. 205-245.

[4] J.C.M. BAETEN, J.A. BERGSTRA & J.W. KLOP (1986): *Syntax and Defining Equations for an Interrupt Mechanism in Process Algebra.* Fund. Inf. IX(2), pp. 127-168.

[5] J.C.M. BAETEN & R.J. VAN GLABBEEK (1987): *Merge and termination in process algebra.* In: Proceedings 7th Conference on Foundations of Software Technology & Theoretical Computer Science, Pune, India (K.V. Nori, ed.), LNCS 287, Springer-Verlag, pp. 153-172.

[6] J.W. DE BAKKER & J.N. KOK (1988): *Uniform Abstraction, Atomicity and Contractions in the Comparative Semantics of Concurrent Prolog.* Report CS-R8834, Centrum voor Wiskunde en Informatica, Amsterdam, to appear in: Proceedings Fifth Generation Computer Systems 1988 (FGCS'88), December 1988, Tokyo, Japan.

[7] J.A. BERGSTRA & J.W. KLOP (1988): *A Complete Inference System for Regular Processes with silent moves.* In: Proceedings Logic Colloquium 1986 (F.R. Drake & J.K. Truss, eds.), North Holland, Hull, pp. 21-81, also appeared as: Report CS-R8420, Centrum voor Wiskunde en Informatica, Amsterdam 1984.

[8] J.A. BERGSTRA, J.W. KLOP & E.-R. OLDEROG (1987): *Readies and Failures in the Algebra of Communicating Processes (revised version).* Report CS-R8748, Centrum voor Wiskunde en Informatica, Amsterdam, to appear in: SIAM Journal of Computing. This paper is a revised version of CWI Report CS-R8523, Amsterdam 1985.

[9] B. BLOOM (November 1988): *Personal communication.*

[10] B. BLOOM, S. ISTRAIL & A.R. MEYER (1988): *Bisimulation Can't Be Traced: Preliminary Report.* In: Proceedings of the Fifteenth POPL, San Diego, California, pp. 229-239.

[11] G. BOUDOL (1985): *Notes on Algebraic Calculi of Processes.* In: Logics and Models of Concurrent Systems (K. Apt, ed.), NATO ASI Series F13, Springer-Verlag, pp. 261-303.

[12] R. DE NICOLA & M. HENNESSY (1984): *Testing equivalences for processes.* Theoretical Computer Science 34, pp. 83-134.

[13] R.J. VAN GLABBEEK (1987): *Bounded Nondeterminism and the Approximation Induction Principle in Process Algebra.* In: Proceedings STACS 87 (F.J. Brandenburg, G. Vidal-Naquet & M. Wirsing, eds.), LNCS 247, Springer-Verlag, pp. 336-347.

[14] R.J. VAN GLABBEEK (November 1988): *Personal communication.*

[15] R.J. VAN GLABBEEK & F.W. VAANDRAGER (1987): *Petri net models for algebraic theories of concurrency.* In: Proceedings PARLE conference, Eindhoven, Vol. II (Parallel Languages) (J.W. de Bakker, A.J. Nijman & P.C. Treleaven, eds.), LNCS 259, Springer-Verlag, pp. 224-242.

[16] M. HENNESSY (1988): *Algebraic Theory of Processes*, MIT Press, Cambridge, Massachusetts.
[17] M. HENNESSY & R. MILNER (1985): *Algebraic laws for nondeterminism and concurrency*. JACM 32(1), pp. 137-161.
[18] R.M. KELLER (1976): *Formal verification of parallel programs*. Communications of the ACM 19(7), pp. 371-384.
[19] J.W. KLOP (1987): *Term Rewriting Systems: A Tutorial*. Bulletin of the EATCS 32, pp. 143-182.
[20] K.G. LARSEN & A. SKOU (1988): *Bisimulation through probabilistic testing*. R 88-29, Institut for Elektroniske Systemer, Afdeling for Matematik og Datalogi, Aalborg Universitetscenter.
[21] R. MILNER (1980): *A Calculus of Communicating Systems*, LNCS 92, Springer-Verlag.
[22] R. MILNER (1981): *Modal characterisation of observable machine behaviour*. In: Proceedings CAAP 81 (G. Astesiano & C. Bohm, eds.), LNCS 112, Springer-Verlag, pp. 25-34.
[23] R. MILNER (1983): *Calculi for synchrony and asynchrony*. Theoretical Computer Science 25, pp. 267-310.
[24] E.-R. OLDEROG & C.A.R. HOARE (1986): *Specification-Oriented Semantics for Communicating Processes*. Acta Informatica 23, pp. 9-66.
[25] D.M.R. PARK (1981): *Concurrency and automata on infinite sequences*. In: Proceedings 5th GI Conference (P. Deussen, ed.), LNCS 104, Springer-Verlag, pp. 167-183.
[26] G.D. PLOTKIN (1981): *A Structural approach to operational semantics*. Technical Report DAIMI FN-19, Computer Science Department, Aarhus University.
[27] G.D. PLOTKIN (1983): *An operational semantics for CSP*. In: Proceedings IFIP TC2 Working Conference on Formal Description of Programming Concepts - II, Garmisch, 1982 (D. Bjørner, ed.), North-Holland, Amsterdam, pp. 199-225.
[28] A. PNUELI (1985): *Linear and Branching Structures in the Semantics and Logics of Reactive Systems*. In: Proceedings 12th ICALP, Nafplion (W. Brauer, ed.), LNCS 194, Springer-Verlag, pp. 15-32.
[29] R. DE SIMONE (1984): *Calculabilité et expressivité dans l'algebra de processus parallèles Meije*. Thèse de 3^e cycle, Univ. Paris 7.
[30] R. DE SIMONE (1985): *Higher-level synchronising devices in* MEIJE-SCCS. Theoretical Computer Science 37, pp. 245-267.
[31] J.L.M. VRANCKEN (1986): *The Algebra of Communicating Processes with Empty Process*. Report FVI 86-01, Dept. of Computer Science, University of Amsterdam.