INFORMATION AND COMPUTATION 100, 202–260 (1992)

Structured Operational Semantics and Bisimulation as a Congruence*

JAN FRISO GROOTE AND FRITS VAANDRAGER[†]

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

In this paper we are interested in general properties of classes of transition system specifications in Plotkin style. 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 conditional rules over the syntax. It is argued that in this setting it is natural to require that strong bisimulation equivalence be a congruence on the states of the transition systems. A general format, called the *tyft/tyxt* format, is presented for the rules in a transition system specification, such that bisimulation is always a congruence when all the rules fit 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 modularization. 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 (*Theoret. Comput. Sci.* **37**, 245–267 (1985)) but is incomparable to the GSOS format of Bloom, Istrail, and Meyer (*in* "Conference Record of the 15th Annual Symposium on Principles of Programming Languages, San Diego, California, 1988," pp. 229–239). However, it turns out that 2-nested simulation equivalence strictly refines the completed trace congruence induced by the GSOS format. © 1992 Academic Press, Inc.

1. INTRODUCTION

Plotkin (1981, 1983) 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

* This paper is a revised version of Report CS-R8845, Centrum voor Wiskunde en Informatica Amsterdam. An extended abstract appeared in "Proceedings ICALP 89, Stresa" (G. Ausiello, M. Dezani-Ciancaglini, and S. Ronchi Della Rocca, Eds.), Lecture Notes in Computer Science, Vol. 372, pp. 423–438, Springer-Verlag, Berlin/New York.

[†] The research of the authors was supported by ESPRIT Project 432, An Integrated Formal Approach to Industrial Software Development (METEOR), and by RACE Project 1046, Specification and Programming Environment for Communication Software (SPECS).

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 to define the transitions between states by what we call a *Transition System Specification* (*TSS*): a set of conditional rules over the syntax of the language.

In recent years a large number of (concurrent) languages have been provided with an operational semantics using Plotkin's approach. Therefore it might be worthwhile to develop a *general* theory of structured operational semantics: to establish a hierarchy of "formats" of transition system specifications and to investigate the expressiveness and properties of each format. We think that it is possible to develop such a general theory: many important properties of transition system specifications can be derived by just looking at the syntactic form of the rules. A general theory of SOS will be useful for several reasons. First, certain results will become reusable so that one does not have to prove them for each individual language separately. Second, a general theory of SOS may lead to a better understanding of the relations between languages that have been provided with a semantics using the approach. Third, one may hope that a general theory helps people in giving good operational semantics: if one knows that certain types of rules have bad properties, then one will try not to use them. Surprisingly, there are not many papers that contain general results on SOS. We are only aware of the work of de Simone (1984, 1985) and Bloom, Istrail, and Meyer (1988).

The aim of this paper is to contribute to the general theory of structured operational semantics. We start from the requirement that strong bisimulaton equivalence should be a congruence for the operators in a transition system specification. We then show how this requirement leads naturally to a certain format of rules, which we call the *tyft*/*tyxt* format. Next we analyze the properties of the *tyft*/*tyxt* format and make comparisons with related work.

In order to facilitate analysis, 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 (Milner, 1980), TCSP (Olderog and Hoare, 1986), ACP (van Glabbeek, 1987), and Meije (Boudol, 1985) has been described in essentially this way. However, there are also many 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 by Plotkin (1983) and the semantics for POOL of America, de Bakker, Kok, and Rutten (1986). We hope that the insights derived from our analysis of a basic case will also be useful in more general settings.

1.1. *Bisimulation as a Congruence.* A fundamental equivalence on the states of a labelled transition system is the strong bisimulation equivalence of Park (1981). Strong bisimulation equivalence seems to be the *finest* extensional behavioural equivalence one would want to impose: it is not clear how two states of a transition system which are strongly bisimilar can 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 the function that computes the transitions associated with a phrase from the transitions associated to its components depends on properties of the transition system which are generally considered to be irrelevant, such as the specific names of states. This function is compositional on the level of (concrete) transition systems but not 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 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 rules have this format (and a small additional requirement is met). 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 (Milner, 1983), (Baeten and van Glabbeek, 1987), and elsewhere. Our experience is that if rules in a TSS do not fit our format, there is a good chance that something is wrong: either bisimulation is not a congruence right away or the congruence property will get lost if more operators and rules are added.

1.2. *Modularity of Transition System Specifications.* Often one wants to add new operators and rules to a TSS. Therefore, a very natural and important operation on TSSs 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, except for certain rules which are not "*pure*," *tyft*/*tyxt* rules behave fine under modularization. Fortunately, nonpure rules are quite pathological and we have never seen an application in which they are used.

1.3. *Trace Congruences.* 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 (De Nicola and Hennessy, 1984). Natural observations that one can make on a process are its (*completed*) *traces*. A *trace* of a process is a finite sequence of actions that can be performed during a run of the process. A trace is *completed* if it leads to a state from where no further actions are possible. 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 first main result of Section 8 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 (Hennessy and Milner, 1985) with no [ ] in the scope of a ⟨ ⟩. Consequently the two trees in Fig. 1, which are not bisimilar, cannot be distinguished by operators defined with pure *tyft*/*tyxt* rules. Also in Section 8, we characterize the

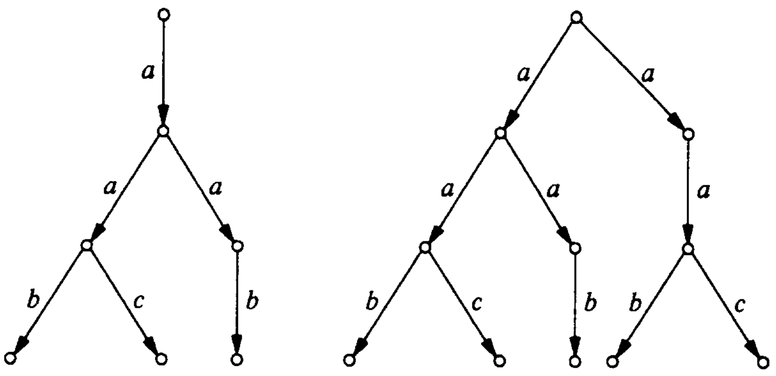


FIG. 1. Pure *tyft*/*tyxt* congruent but not bisimilar.

trace congruence induced by the pure *tyft*/*tyxt* format as *simulation equivalence.*

1.4. *Comparison with Related Work.* In Section 9 we give an extensive comparison of our format with the format proposed by de Simone (1984, 1985) and the GSOS format of Bloom, Istrail, and Meyer (1988). Roughly speaking, the situation is as displayed in Fig. 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 (Bloom, Istrail, and Meyer, 1988) shows that the combination of negation and lookahead is inconsistent in general. The point where the two formats diverge is characterized by the rules which fit the GSOS format but which contain no negation. We call the corresponding format *positive GSOS.*

From results of de Simone (1985) and Bergstra, Klop, ad Olderog (1988) it follows that the completed trace congruence that corresponds to the format of de Simone coincides with *failure equivalence.* Bloom, Istrail, and Meyer (1988) 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 occur in the scope of a [ ]. Larsen and Skou (1989) in turn showed that the equivalence induced by this class of logical formulas can be characterized as $\frac{2}{3}$*-bisimulation equivalence.* From these results we can conclude quite directly that the pure *tyft*/*tyxt* format can make more distinctions between processes than the GSOS format: 2-nested simulation refines $\frac{2}{3}$-bisimulation. Now, interestingly, it turns out that the completed trace congruence induced by the *positive* GSOS format is also $\frac{2}{3}$-bisimulation equivalence. So although it may be the case that the general

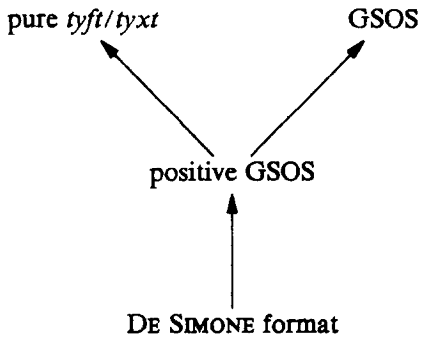


FIGURE 2

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 symbol $*$ denoting concatenation, and constant symbols λ (empty string) and $\mathrm{a}, \mathrm{b}, ..., \mathrm{y}, \mathrm{z}$. As alphabet we take $A = \{\mathrm{a}, \mathrm{b}, ..., \mathrm{y}, \mathrm{z}, \Delta\}$. Here, Δ stands for a backspace. Rules for the typewriter can be given as follows:

$$\begin{array}{rl} x \xrightarrow{a} x * a & \text{for } \ a \in \{\mathrm{a}, \mathrm{b}, ..., \mathrm{y}, \mathrm{z}\} \\ a \xrightarrow{\Delta} \lambda & \text{for } \ a \in \{\mathrm{a}, \mathrm{b}, ..., \mathrm{y}, \mathrm{z}\} \\ x * a \xrightarrow{\Delta} x & \text{for } \ a \in \{\mathrm{a}, \mathrm{b}, ..., \mathrm{y}, \mathrm{z}\}. \end{array}$$

This description of the typewriter is not in *tyft/tyxt* format, because the lhs of the last axiom contains two function symbols. 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:

$$\begin{array}{rl} x \xrightarrow{a} x * a & \text{for } \ a \in \{\mathrm{a}, \mathrm{b}, ..., \mathrm{y}, \mathrm{z}\} \\ a \xrightarrow{\Delta} \lambda & \text{for } \ a \in \{\mathrm{a}, \mathrm{b}, ..., \mathrm{y}, \mathrm{z}\} \\ \lambda \xrightarrow{\text{empty}} \lambda & \\ \dfrac{x \xrightarrow{\Delta} x'}{y * x \xrightarrow{\Delta} y * x'} & \\ \dfrac{x \xrightarrow{e} x' \quad y \xrightarrow{\text{empty}} y'}{x * y \xrightarrow{e} x'} & \text{for } \ e \in \{\text{empty}, \Delta\}. \end{array}$$

We come back to this example in Section 5.11.2.

5.4. *Well-Foundedness.* 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 circular reference. In general y_1 will 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 exclude this type of dependencies, as they give rise to complicated TSSs. 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:

2.3. Definition. Let $\Sigma = (F, r)$ be a signature. A *substitution* σ is a mapping in $V \to \mathbb{T}(\Sigma)$. A substitution σ is extended to a mapping $\sigma\colon \mathbb{T}(\Sigma) \to \mathbb{T}(\Sigma)$ 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 $t_1, ..., t_{r(f)} \in \mathbb{T}(\Sigma)$.

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

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

2.4. *Note.* Observe that we have the following identities:

$$\begin{aligned} \sigma \circ \rho(t) &= \sigma(\rho(t)) && t \in \mathbb{T}(\Sigma) \\ \sigma(t) &= t && \text{for} \quad t \in T(\Sigma). \end{aligned}$$

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 triple (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 an index set, t_i, t_i', t, $t' \in \mathbb{T}(\Sigma)$, and $a_i, a \in A$ for $i \in I$. If r is a rule in the format above, then the elements of $\{t_i \xrightarrow{a_i} t_i' \mid i \in I\}$ are called the *premises* or *hypotheses* 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 $t, t' \in \mathbb{T}(\Sigma)$ is called a *transition* (*labelled with* a). The symbols $\phi, \psi, \chi, ...$ are 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 well-founded, upwardly branching tree of which the nodes are labelled by transitions $t \xrightarrow{a} t'$ with $t, t' \in \mathbb{T}(\Sigma)$ 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 $\sigma\colon V \to \mathbb{T}(\Sigma)$ 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 $\sigma\colon V \to T(\Sigma)$ by $\sigma(x) = t$ for all $x \in V$ (in fact, any closed term will do). 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 $t \xrightarrow{a} t'$. ∎

TSSs 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 $\Sigma(\mathrm{BPA}_\delta^\varepsilon)$ (Basic Process Algebra with δ and ε) as introduced in Vrancken (1986). $\Sigma(\mathrm{BPA}_\delta^\varepsilon)$ contains constants a for each $a \in Act$, a constant δ that stands for *deadlock* or *inaction*, 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 skip in 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 that execution has finished.

Define the TSS $P(\mathrm{BPA}_\delta^\varepsilon)$ as $(\Sigma(\mathrm{BPA}_\delta^\varepsilon), Act_{\surd}, R(\mathrm{BPA}_\delta^\varepsilon))$ where $R(\mathrm{BPA}_\delta^\varepsilon)$ is defined in Table 1. In the table a ranges over $Act_{\surd}$, unless further restrictions are made. Infix notation is used for the binary function symbols.

TABLE 1

The Rules of $R(\mathrm{BPA}_{\delta}^{\varepsilon})$

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.	$\dfrac{x \xrightarrow{\surd} x' \quad y \xrightarrow{a} y'}{x \cdot y \xrightarrow{a} y'}$

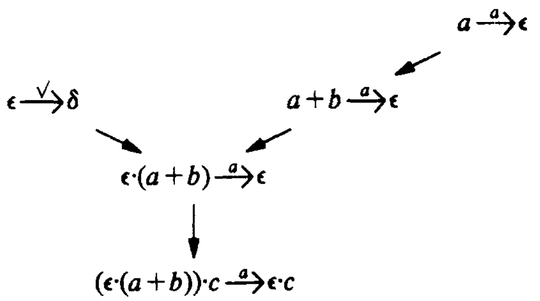
One can easily check that the tree in Fig. 3 constitutes a proof of the transition $(\varepsilon \cdot (a+b)) \cdot c \xrightarrow{a} \varepsilon \cdot c$ from $P(\mathrm{BPA}_{\delta}^{\varepsilon})$.

3.4.1. *Remark.* Even though similar semantic interpretations have been given to (extensions of) $\Sigma(\mathrm{BPA}_{\delta}^{\varepsilon})$ at a number of places, the rules of Table 1 seem to be new. Vrancken (1986) does not use inductive rules to give semantics to $\mathrm{BPA}_{\delta}^{\varepsilon}$. Instead, operations are defined directly on *process graphs*. In (Baeten and van Glabbeek, 1987) there are no transitions labelled with $\surd$. Instead, a unary termination predicate $\downarrow$ is used. 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 the framework of this paper. We have chosen not to deal here with predicates like $\downarrow$ because the additional complexity would distract attention from the main issues in this paper. Moreover, a unary predicate $p(x)$ can always be coded in our setting by adding a new label a_p and rules such that

$$p(x) \Leftrightarrow \exists y\colon x \xrightarrow{a_p} y.$$



$$\begin{tikzcd}
 & & a \xrightarrow{a} \epsilon \arrow[dl] \\
\epsilon \xrightarrow{\surd} \delta \arrow[dr] & a + b \xrightarrow{a} \epsilon \arrow[d] & \\
 & \epsilon \cdot (a+b) \xrightarrow{a} \epsilon \arrow[d] & \\
 & (\epsilon \cdot (a+b)) \cdot c \xrightarrow{a} \epsilon \cdot c &
\end{tikzcd}$$

FIGURE 3

We think that it will not be too difficult to extend the framework of this paper with predicates.

3.5. Example. Our next example shows that the range of applications of TSSs is not restricted to the area of operational semantics: every Term Rewriting System (TRS) can be viewed as a TSS. 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 $t, s \in \mathbb{T}(\Sigma_0)$. Here, t contains at least one function symbol and $Var(s) \subseteq Var(t)$.

A TRS (Σ_0, R_0) 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 symbol 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 Klop, 1987) for a definition) iff the corresponding TSS proves $t \xrightarrow{r} s$.

Apparently, the intersection of the class of TSSs which correspond to TRSs and the class of TSSs for which it is proved in Theorem 5.10 that bisimulation is a congruence is of no interest: Theorem 5.10 requires that not more than one function symbol occur in the source of an axiom.

4. Transition Systems and Strong Bisimulation Equivalence

An operational semantics makes use of some sort of (abstract) machines 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 (Keller, 1976).

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 are 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 some small technical advantages are gained by working 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 (1981).

4.2. Definition. Let $\mathcal{A} = (S, A, \rightarrow)$ be a labelled transition system. A relation $R \subseteq S \times S$ is a (*strong*) *bisimulation* if for all s, t with $s\ R\ t$:

1. Whenever $s \xrightarrow{a} s'$ for some a and s', then, for some t', also $t \xrightarrow{a} t'$ and $s'\ R\ t'$.

2. Conversely, whenever $t \xrightarrow{a} t'$ for some a and t', then, for some s', also $s \xrightarrow{a} s'$ and $s'\ R\ t'$.

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

4.3. Definition (TSSs, transition systems, and bisimulation). Let $P = (\Sigma, A, R)$ be a TSS. The transition system $TS(P)$ specified by P is given by

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

where relation $\rightarrow_P \subseteq T(\Sigma) \times A \times T(\Sigma)$ is defined by $t \xrightarrow{a}_P t' \Leftrightarrow P \vdash t \xrightarrow{a} t'$.

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

4.4. Example. For the TSS $P(\text{BPA}_\delta^\varepsilon)$ 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. As in regular algebra we often omit the $\cdot$ in a product $x \cdot y$ and we take $\cdot$ to be more binding than $+$.

(a) $\varepsilon\varepsilon \leftrightarroweq \varepsilon$

(b) $b \leftrightarroweq b + b$

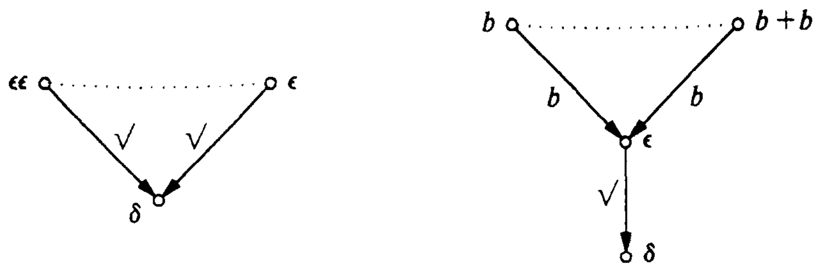


FIG. 4. Examples 4.4(a) and (b).

(c) $(\varepsilon a+\varepsilon b)(c(d\delta)+\delta) \leftrightarrows (a((c+\delta)d)+b(c(d+d)))\delta$

(d) $b\varepsilon \leftrightarrows b$

(e) $\varepsilon b \leftrightarrows b$

(f) $ab+ac \not\leftrightarrows a(b+c)$

The parts of the automaton belonging to (a), (b), (c), and (f) are drawn in Figs. 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. In Figs. 5 and 6 two separate automata are drawn instead of a combined one, to make the pictures clearer.

In Fig. 6 the states $a(b+c)$ and $\varepsilon(b+c)$ in the right transition system cannot be related to any of the states in the left transition system.

5. Compositional Transition System Specifications

TSSs do not always generate automata for which strong bisimulation is a congruence. A number of examples follow in the sequel. But if the

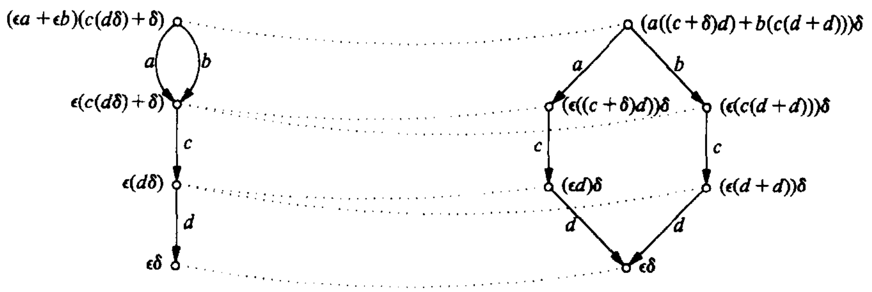


FIG. 5. Example 4.4(c).

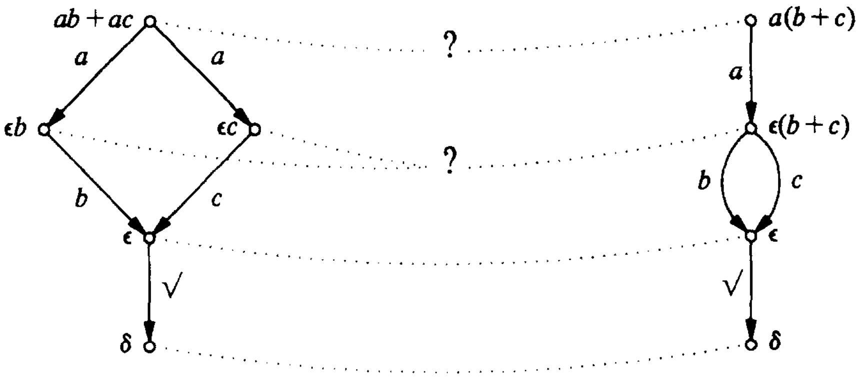


FIG. 6. Example 4.4(f).

rules in TSS satisfy the format below (and an additional small technical requirement is met), strong bisimulation turns 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 form

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

with I an index set, $f \in F$, $r(f)=n$, x_i $(1 \leqslant i \leqslant n)$ and y_i $(i \in I)$ all different variables from V, a_i, $a \in A$, and t_i, $t \in \mathbb{T}(\Sigma)$ for $i \in I$.

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

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

with I an index set, x, y_i $(i \in I)$ all different variables from V, a_i, $a \in A$, and t_i, $t \in \mathbb{T}(\Sigma)$ for $i \in I$. P is in *tyft/tyxt format* if every rule in R is either in *tyft* format or in *tyxt* format. A transition system $\mathcal{A}$ is called *tyft/tyxt specifiable* if there exists a TSS P in *tyft/tyxt* format with $\mathcal{A} = TS(P)$.

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 $P(\mathrm{BPA}_\delta^\varepsilon)$ of Example 3.4 in *tyft/tyxt* format, but also any TSS obtained from $P(\mathrm{BPA}_\delta^\varepsilon)$ by dropping some 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 symbol $*$ denoting concatenation, and constant symbols λ (empty string) and $\mathrm{a}, \mathrm{b}, ..., \mathrm{y}, \mathrm{z}$. As alphabet we take $A = \{\mathrm{a}, \mathrm{b}, ..., \mathrm{y}, \mathrm{z}, \Delta\}$. Here, Δ stands for a backspace. Rules for the typewriter can be given as follows:

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

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

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

This description of the typewriter is not in *tyft/tyxt* format, because the lhs of the last axiom contains two function symbols. 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:

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

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

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

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

$$\frac{x \xrightarrow{e} x' \quad y \xrightarrow{\text{empty}} y'}{x * y \xrightarrow{e} x'} \qquad \text{for} \quad e \in \{\text{empty}, \Delta\}.$$

We come back to this example in Section 5.11.2.

5.4. *Well-Foundedness.* 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 circular reference. In general y_1 will 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 exclude this type of dependencies, as they give rise to complicated TSSs. 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: $\{\langle x, y \rangle \mid x \in Var(t_i),\ y \in Var(t'_i)$ for some $i \in I\}$.

A set of transitions is called *well-founded* if any backward chain of edges in the dependency graph of these transitions is finite. A rule is called *well-founded* if the set of its premises is so. Finally, a TSS is called *well-founded* if all its rules are well-founded.

5.4.2. EXAMPLE. The dependency graph of the set of premises of the rule in Section 5.4 is given in Fig. 7. The rule is not well-founded since the graph clearly contains a cycle.

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

Hence, two TSSs are transition 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

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

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

To deal with closed terms, only the *tyft* format is necessary and the *tyxt* format is not needed:

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

Proof. Let $\Sigma = (F, \text{rank})$. Define R' by:

— every *tyft* rule of R is in R',

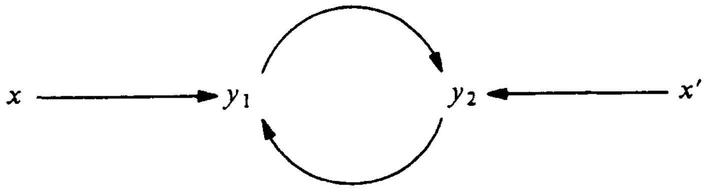


FIGURE 7

— for every *tyxt* rule $r \in R$ and for every function symbol $f \in F$, r_f is in R', where r_f is obtained by substituting $f(x_1, ..., x_{\mathrm{rank}(f)})$ for x in r with $\{x_1, ..., x_{\mathrm{rank}(f)}\} \subseteq V - Var(r)$.

If the old *tyxt* rules were well-founded, then the new rules will be well-founded too and in *tyft* format. Suppose that $t \xrightarrow{a} t'$ is a 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 well-founded 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 well-founded TSS in tyft/tyxt format. Then there is a transition 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 well-founded 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 well-founded 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 well-founded and in tyft/tyxt format then strong bisimulation is a congruence for all function symbols; i.e., for all function symbols f in F and all closed terms $u_i, v_i \in T(\Sigma)$ $(1 \leqslant i \leqslant r(f))$,*

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

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 be 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 well-founded cannot be missed: we just have not been able to prove the theorem without it. However, non-well-founded TSSs are quite pathological and we know of no application. In Section 7 it will be shown that non-well-founded rules are ill-behaved with respect to modularization.

5.11. COUNTEREXAMPLES.

5.11.1. EXAMPLE. The first example shows that in general the variables in the source of 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 $P(\mathrm{BPA}_\delta^\varepsilon)$ the axiom $x+x \xrightarrow{ok} \delta$. We then have $a \leftrightarroweq a\varepsilon$, but $a+a \not\leftrightarroweq a+a\varepsilon$ as a and $a\varepsilon$ are not syntactically equal.

5.11.2. EXAMPLE. In general, not more than one function may occur in the source of the conclusion. Take the TSS $P(\mathrm{BPA}_\delta^\varepsilon)$ extended with the axiom $ab \xrightarrow{ok} \delta$. As in Example 4.4(b) $b \leftrightarroweq b+b$, but in the new situation we do not have any more that $ab \leftrightarroweq a(b+b)$ 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 $P(\mathrm{BPA}_\delta^\varepsilon)$. Again we have $b \leftrightarroweq b+b$, but now it is not the case that $b+(b+b) \leftrightarroweq b+b$.

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 symbols. Now $\lambda * a \leftrightarroweq a$ but $a * (\lambda * a) \not\leftrightarroweq a * a$. Bisimulation *is* a congruence for the *tyft*/*tyxt* version of the typewriter. The reader may also check that the identities $\lambda * t \leftrightarroweq t * \lambda \leftrightarroweq t$ and $(s * t) * u \leftrightarroweq s * (t * u)$ with s, t, u closed terms over the signature hold for the second version of the typewriter but not for the first version.

5.11.3. EXAMPLE. Our next example shows that on the right hand side of a premise, function symbols are not allowed to occur. We can add *prefixing* operators $a\colon(\cdot)$ to $P(\mathrm{BPA}_\delta^\varepsilon)$ for each $a \in Act$ and define the operational meaning of these operators with rules:

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

If we now add moreover the rule

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

we have problems because $a{:}a{:}\delta \not\leftrightarroweq a{:}a{:}(\delta+\delta)$ even though $\delta \leftrightarroweq \delta+\delta$.

5.11.4. Example. The variables in the right hand sides of the arrows in the premises must in general be different. This is shown by adding the rule

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

to $P(\mathrm{BPA}_\delta^\varepsilon)$. Now $a \leftrightarrow a\varepsilon$, but $aa \not\leftrightarrow (a\varepsilon)a$.

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 $P(\mathrm{BPA}_\delta^\varepsilon)$ and observe that $\varepsilon\varepsilon \leftrightarrow \varepsilon$, but $a + \varepsilon\varepsilon \not\leftrightarrow a + \varepsilon$.

5.12. We now prove Theorem 5.10.

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

— $\leftrightarrow_P \subseteq R$,

— for all function symbols f in F and terms u_i, v_i $(1 \leqslant i \leqslant r(f))$ in $T(\Sigma)$,

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

It is enough to show that $R \subseteq \leftrightarrow_P$ because then $R = \leftrightarrow_P$ and it follows from the definition of R that $\leftrightarrow_P$ is a congruence for all f in F. In order to prove that $R \subseteq \leftrightarrow_P$ 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 $u \leftrightarrow_P v$ or, for some function symbol 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 $\leftrightarrow_P$ 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 T of $f(u_1, ..., u_{r(f)}) \xrightarrow{a} u'$ that contains only closed transitions. We prove the statement with ordinal induction on the structure of T. Lemma 5.9 allows us to assume throughout the proof that the rules in R_0 are pure and in *tyft* format.

Let r be the last rule used in proof T, in combination with a substitution σ. 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'. Consider the dependency graph G of the premises of r. Because r is *tyft*, each node in G has at most finitely many incoming edges. Because G is well-founded we can define for each node x of G, depth$(x) \in \mathbb{N}$ as the length of the maximal backward chain of edges (use König's lemma). Define

$$X = \{x_i \mid 1 \leqslant i \leqslant r(f)\}$$
$$Y = \{y_i \mid i \in I\}$$
$$Y_n = \{y \in Y \mid \text{depth}(y) = n\} \qquad \text{for} \quad n \geqslant 0.$$

Observe that for any variable $x \in X$, depth$(x) = 0$, and that the sets Y_n form a partition of Y. We will define a substitution σ' that satisfies the following properties:

$$\sigma'(x_i) = v_i \qquad \text{for} \quad 1 \leqslant i \leqslant r(f) \tag{1}$$
$$\sigma(y)\, R\, \sigma'(y) \qquad \text{for} \quad y \in X \cup Y \tag{2}$$
$$P \vdash \sigma'(t_i \xrightarrow{a_i} y_i) \qquad \text{for} \quad i \in I. \tag{3}$$

Substitution σ' will be constructed in a stepwise fashion. To begin with we define

$$\sigma'(x_i) = v_i \qquad \text{for} \quad 1 \leqslant i \leqslant r(f)$$
$$\sigma'(y) = \sigma(y) \qquad \text{for} \quad y \in V - \left(X \cup \bigcup_{n>0} Y_n\right)$$

We still have to define σ' on $\bigcup_{n>0} Y_n$. As soon as σ' has been defined for all variables in $X \cup Y_0 \cup \cdots \cup Y_m$ $(m \geqslant 0)$, we can state the following properties $\alpha(m)$ and $\beta(m)$:

$$\alpha(m)\colon \sigma(y)\, R\, \sigma'(y) \qquad \text{for} \quad y \in X \cup Y_0 \cup \cdots \cup Y_m$$
$$\beta(m)\colon P \vdash \sigma'(t_i \xrightarrow{a_i} y_i) \qquad \text{for} \quad y_i \in Y_0 \cup \cdots \cup Y_m.$$

One can easily check that $\alpha(0)$ and $\beta(0)$. Let $n>0$. Suppose that σ' has been defined already for all variables in $X \cup Y_0 \cup \cdots \cup Y_{n-1}$ in such a way that properties $\alpha(n-1)$ and $\beta(n-1)$ hold. We show how to define σ' on all variables of Y_n such that $\alpha(n)$ and $\beta(n)$ hold. This is sufficient for completing the definition of a σ' that satisfies properties 1–3: property 1 is met by definition, properties 2 and 3 follow because σ' satisfies properties $\alpha(n)$, resp. $\beta(n)$, for all $n \in \mathbb{N}$.

Pick an element $y^* \in Y_n$. There is a unique $i \in I$ with $y^* = y_i$. Because $y_i \in Y_n$ and rule r is pure, $\mathrm{Var}(t_i) \subseteq X \cup Y_0 \cup \cdots \cup Y_{n-1}$. Now use that σ' satisfies $\alpha(n-1)$ to obtain that for all variables $y \in \mathrm{Var}(t_i)$: $\sigma(y)\, R\, \sigma'(y)$. Next we use the following

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

Proof. Straightforward induction on the structure of term t using the definition of R. ▌

We obtain that $\sigma(t_i)\, R\, \sigma'(t_i)$. Since also $P \vdash \sigma(t_i) \xrightarrow{a_i} \sigma(y_i)$, we can distinguish, by definition of R, between two cases:

(1) $\sigma(t_i) \leftrightarrows_P \sigma'(t_i)$. In this case we can find a $w \in T(\Sigma)$ such that $P \vdash \sigma'(t_i) \xrightarrow{a_i} w$ and $\sigma(y_i)\, R\, w$. We then define $\sigma'(y^*) = \sigma'(y_i) = w$.

(2) There is a function symbol g in F and there are terms w_j, w'_j for $1 \leqslant j \leqslant r(g)$ such that

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

$$\sigma'(t_i) = g(w'_1, ..., w'_{r(g)}),$$

and

$$w_j\, R\, w'_j \qquad \text{for} \quad 1 \leqslant j \leqslant r(g).$$

But now we can apply the induction hypothesis which gives that we can find a w such that $P \vdash g(w'_1, ..., w'_{r(g)}) \xrightarrow{a_i} w$ and $\sigma(y_i)\, R\, w$. We define $\sigma'(y^*) = \sigma'(y_i) = w$.

In the same way we can define σ' for the other elements of Y_n. It is not hard to see that after this $\alpha(n)$ and $\beta(n)$ hold.

Let for $i \in I$, T_i be a proof of $\sigma'(t_i \xrightarrow{a_i} y_i)$. Construct a proof T' with root $\sigma'(f(x_1, ..., x_{r(f)}) \xrightarrow{a} t)$ and as direct subtrees the proofs T_i ($i \in I$). Define $v' = \sigma'(t)$. Clearly T' is a proof for $f(v_1, ..., v_{r(f)}) \xrightarrow{a} v'$. Since for all $x \in \mathrm{Var}(t)$, $\sigma(x)\, R\, \sigma'(x)$ (use that r is pure), it follows by an application of the previously stated fact that $\sigma(t)\, R\, \sigma'(t)$ or, equivalently, $u'\, R\, v'$. ▌

5.13. The implication in Theorem 5.10 cannot be reversed. So given a TSS for which bisimulation is a congruence, this TSS need not be well-founded and in *tyft*/*tyxt* format. This is obvious because for any TSS, a transition equivalent TSS can be obtained by adding all derivable transitions as rules. And if bisimulation is a congruence for the one it is a congruence for the other. If one starts from a well-founded TSS in *tyft*/*tyxt* format, the result will in general not be *tyft*/*tyxt*. For instance, in the case of $P(\mathrm{BPA}_\delta^\varepsilon)$ one adds the rule $a \cdot (x+y) \xrightarrow{a} \varepsilon \cdot (x+y)$.

Even after derivable rules are removed, a TSS for which bisimulation is a congruence need not be well-founded and in *tyft*/*tyxt* format. The TSS $P'(\mathrm{BPA}_\delta^\varepsilon)$ described in Section 5.5 contains no derivable rules and is not in *tyft*/*tyxt* format. But, as observed in that section, it is transition equivalent to the TSS $P(\mathrm{BPA}_\delta^\varepsilon)$ which is in *tyft*/*tyxt* format. Hence, bisimulation equivalence is a congruence.

It is worth noting that if one adds new operators and rules to $P'(\mathrm{BPA}_\delta^\varepsilon)$, the congruence property can get lost, even if the rules for the new operators are *tyft*. In order to see this, consider the TSS obtained by adding to $P'(\mathrm{BPA}_\delta^\varepsilon)$ *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 $a \leftrightarrow \partial_{\{b\}}(a)$, but $a \cdot b \not\leftrightarrow \partial_{\{b\}}(a) \cdot b$.

The examples above do not rule out the following weakened variant of the reverse implication of Theorem 5.10: if P is a TSS for which bisimulation is a congruence, then $TS(P)$ can be specified by a well-founded TSS in *tyft*/*tyxt* format. Below we present a TSS that eliminates this variant of the reverse implication. Consider the TSS P that has constant symbols a, b, and δ, a binary function symbol f, labels a, b, 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 *tyft*/*tyxt*, but it is not hard to see that $\leftrightarrow_P$ is a congruence. We claim that there exists no TSS in *tyft*/*tyxt* format that is transition equivalent to P. In order to prove this claim, it is, by Lemma 5.6 and the proof of Lemma 5.9, sufficient to show that no TSS P' in *tyft* format and without free variables in the rules can be transition equivalent to P. Suppose there were such a P'. Since $P' \vdash f(a) \xrightarrow{c} \delta$, there is a closed proof T of $f(a) \xrightarrow{c} \delta$ such that only the root of T is labelled with $f(a) \xrightarrow{c} \delta$. The other nodes in T are labelled with either $a \xrightarrow{a} \delta$, $b \xrightarrow{a} \delta$, or $b \xrightarrow{b} \delta$. Let r be the last rule used in T, in combination with a substitution σ. Rule r must be of the form

$$\frac{\{t_i \xrightarrow{a_i} t_i' \mid i \in I\}}{f(x) \xrightarrow{c} t}.$$

It is not hard to see that for $i \in I$, t_i must be equal to x, a, or b. Clearly $\sigma(x) = a$. Let σ' be the same as σ except that $\sigma'(x) = b$. Let $i \in I$. Then $\sigma'(t_i \xrightarrow{a_i} t_i')$ is either $a \xrightarrow{a} \delta$, $b \xrightarrow{a} \delta$, or $b \xrightarrow{b} \delta$. Moreover $\sigma'(t) = \delta$. Thus we can construct a proof from P' of transition $f(b) \xrightarrow{c} \delta$ by taking r as a last rule with substitution σ' and appending proofs of $a \xrightarrow{a} \delta$, $b \xrightarrow{a} \delta$ and $b \xrightarrow{b} \delta$ on top of that at the appropriate places. Contradiction.

Also in this case we have that adding *tyft* rules may destroy the congruence property (take the axiom $a \xrightarrow{b} \delta$).

5.14. *Remark.* The examples of Section 5.13 show that there is another reason for using TSSs in *tyft*/*tyxt* format, namely their extensibility without endangering congruence properties. We conjecture that, whenever a TSS contains a non-*tyft*/*tyxt* rule, it is possible to extend this TSS (except for some trivial cases, for instance if the non-*tyft*/*tyxt* rules are derivable) with a number of *tyft* 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 TSSs 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 *observation congruence* (Milner, 1980) and *rooted-τ-bisimulation* (Bergstra and Klop, 1988) have been introduced. As observed by van Glabbeek (1987) it is not necessary to introduce a new notion of bisimulation: one can just work with the standard notion of strong bisimulation if one is willing to add some Plotkin style rules that capture the notion of a hidden, internal machine step.

Below we assume that τ is an element of the set *Act* of actions that figures as a parameter of the TSS $P(\mathrm{BPA}_\delta^\varepsilon)$. The TSS $P(\mathrm{BPA}_{\varepsilon\delta}^\tau)$ is obtained by adding to $P(\mathrm{BPA}_\delta^\varepsilon)$ the rules of Table 2 ($a \in Act_\surd$).

One possible interpretation that one can give to a transition $t \xrightarrow{a} t'$ ($a \neq \tau$) is that the system that is modelled can evolve from state t to state

TABLE 2

Rules for the Silent Move τ

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

t' during a certain positive time interval in which an occurrence of action a can be observed. Then $t \xrightarrow{\tau} t'$ means that no action can be observed during such an interval. Rules 8 and 9 can be viewed as logical consequences of this interpretation. It is consistent with the interpretation of transitions and the rules of Table 1 and Table 2 to assume that execution of a *process* a takes a positive amount of time; the observation of the *action* a, however, takes place at the beginning. Rule 7 says that when the action a is observed, the process a that executes this action may still perform some internal activity before it terminates successfully.

The TSS $P(\mathrm{BPA}^{\tau}_{\varepsilon\delta})$ is in pure *tyft*/*tyxt* format. Thus strong bisimulation is a congruence. One can prove that the theory $\mathrm{BPA}^{\tau}_{\varepsilon\delta}$, as presented in Table 3 (a ranges over Act), is a sound and complete axiomatization of the model induced by the TSS $P(\mathrm{BPA}^{\tau}_{\varepsilon\delta})$ modulo strong (!) bisimulation. This means that, if $\leftrightarrows_{r\tau\delta}$ denotes rooted-τ-bisimulation (i.e., observation congruence), we have the following situation:

$$P(\mathrm{BPA}^{\tau}_{\varepsilon\delta}) \models s \leftrightarrows t \Leftrightarrow P(\mathrm{BPA}^{\varepsilon}_{\delta}) \models s \leftrightarrows_{r\tau\delta} t \Leftrightarrow \mathrm{BPA}^{\tau}_{\varepsilon\delta} \vdash s = t.$$

TABLE 3

The Axiom System $\mathrm{BPA}^{\tau}_{\varepsilon\delta}$

$x + y = y + x$	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		

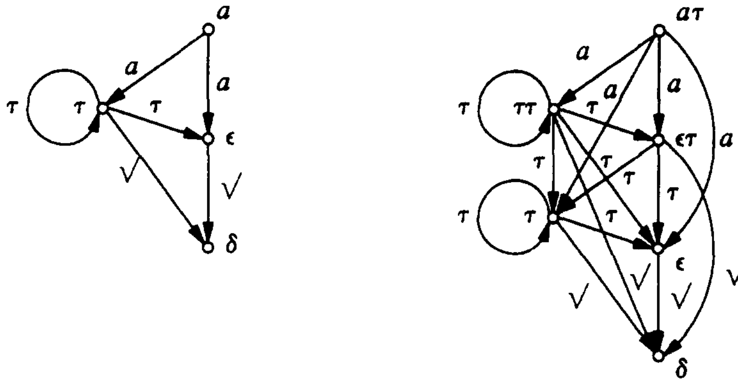


FIG. 8. $(a = a\tau)$.

In Figs. 8–10 we give three examples corresponding to the τ-laws of Milner (1980). In Fig. 8 two separate transition systems are drawn. In Figs. 8 and 10 a may not equal τ. In Fig. 9 the relevant states of $\tau + \varepsilon$ and τ are drawn, as the equation $\tau + \varepsilon = \tau$ is equivalent to the axiom T2. It is left to the reader to check that the transition systems are strongly bisimilar.

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 constant symbols. 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

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

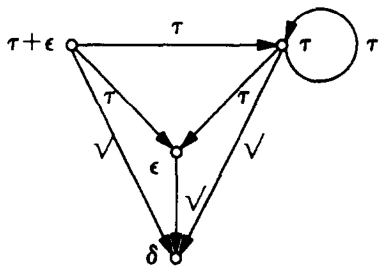


FIG. 9. $(\tau + \varepsilon = \tau)$.

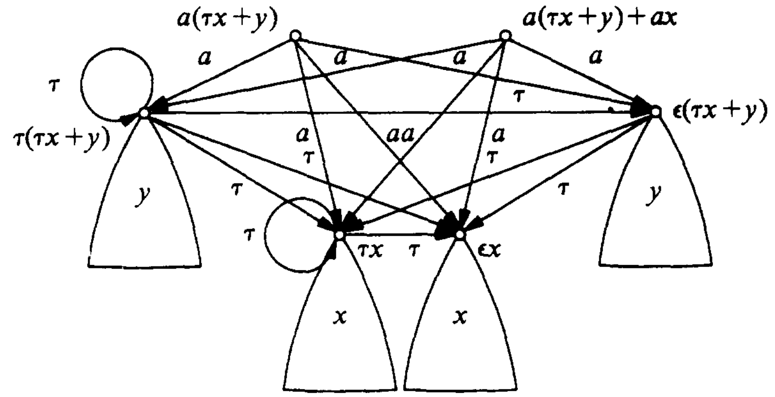


FIG. 10. $(a(\tau x+y)=a(\tau x+y)+ax)$.

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

A slightly different way of dealing with recursion is followed by Olderog and Hoare (1986) and Hennessy (1988). Here axioms $X \xrightarrow{\tau} t_X$ appear saying that by some internal activity, a process name can expand to its body. This type of rules also satisfies our format.

6.3. *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 (America *et al.*, 1986; Plotkin, 1983)). Here, states of the transition system are generally configurations, i.e., pairs $\langle t, \sigma \rangle$ 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 within the *tyft*/*tyxt* format using the *extended state operator* Λ_σ of Baeten and Bergstra (1988).

We add the state operator to the setting of $\mathrm{BPA}^{\tau}_{\varepsilon\delta}$ of Section 6.1. Let S be a set of *states*. For each $\sigma \in S$ we add a function symbol Λ_σ to the signature. An expression $\Lambda_\sigma(t)$ represents a process that transforms the state σ during successive transitions of t as specified by a function effect: $S \times Act \times Act \to S$ while influencing the actual labels of the transitions of t as specified by a function action: $Act \times S \to 2^{Act}$. action(a, σ) defines the set of actions that can be performed by $\Lambda_\sigma(t)$ if t performs an a. effect(σ, a, b) defines the resulting state if $\Lambda_\sigma(t)$ actually transforms under $b \in$ action(a, σ). 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 may determine which action from this set actually will occur. The functions effect and action are *inert* for τ; i.e., action$(\tau, \sigma) = \{\tau\}$ and effect$(\sigma, \tau, a) = \sigma$ for every $a \in Act$. The rules for the state operator are ($\sigma \in S$; $a, b \in Act$):

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

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

Clearly the above rules are pure *tyft*, so bisimulation will be a congruence. As a typical application we consider a small subset of CSP. Actions in *Act* are of the form τ, $g!e$, $g?v$, or $[v := e]$, where v ranges over a set $\mathscr{V}$ of program variables and e ranges over natural number expressions built from $\mathscr{V}$, constants for the natural numbers, and the 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 v" and $[v := e]$ means "assign the value of expression e to v." We assume the presence of an interpretation function $[\![\cdot]\!]$ that, given a valuation σ of the variables, yields for each expression a natural number. As state space S we take all valuations in $\mathscr{V} \to \mathbb{N}$. Let $\sigma[n/v]$ be the valuation σ except for the fact that variable v is mapped on n. Now we can define the functions action and effect as follows:

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

Function effect is inert in the cases that are not specified. As an example consider a process that is capable of reading a value from channel g_1 and sending the square of that value to channel g_2:

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

A particular sequence of transitions of this process is:

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

It is not difficult to extend the combination of $\mathrm{BPA}^\tau_{\varepsilon\delta}$ and the state operator with a parallel combinator. Then, communication can be defined such that we have value passing between several processes. We will not give a detailed elaboration of this because that would go beyond the scope of this article. However, we would like to stress that in 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

Often one wants to add new operators and rules to a given TSS. Therefore, a very natural operation on TSSs is to take their componentwise union. Given two TSSs 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

$$\Sigma_0 \oplus \Sigma_1 = (F_0 \cup F_1, \lambda f. \text{if } f \in F_0 \text{ then } r_0(f) \text{ else } r_1(f)).$$

7.2. Definition. Let $P_i = (\Sigma_i, A_i, R_i)$ $(i=0, 1)$ be two TSSs 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 TSSs 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.

7.4. *Remark.* Let $P_i = (\Sigma_i, A_i, R_i)$ $(i=0, 1)$ be two TSSs 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 $s, t \in T(\Sigma_0)$,

$$s \leftrightarroweq_P t \Leftrightarrow s \leftrightarroweq_{P_0} t.$$

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 restrict ourselves to the case where both P_0 and P_1 are in *tyft/tyxt* format. Below, 5 examples are presented that illustrate different situations where we do not have conservativity.

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

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

then $\delta \leftrightarroweq_{P_0} \delta + \delta$ but not $\delta \leftrightarroweq_{P_0 \oplus P_1} \delta + \delta$.

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 constant symbols a, b, a label a, and rules

$$a \xrightarrow{a} a$$

$$\frac{x \xrightarrow{a} y}{b \xrightarrow{a} y}.$$

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

7.5.3. EXAMPLE. Conservativity can get lost also if free variables occur in the conclusion of a rule in P_0. Let the signature of P_0 consists of two constant symbols a and b. The set of labels contains only a and there are two axioms:

$$a \xrightarrow{a} a$$

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

It is not hard to see that $a \leftrightarroweq_{P_0} b$. However, if we add a TSS P_1 which contains a constant symbol c and no rules, then $a \not\leftrightarroweq_{P_0 \oplus P_1} b$.

7.5.4. EXAMPLE. Conservativity up to bisimulation can be violated if we add *tyxt* rules to a given TSS. Let P_0 consist of $P(\mathrm{BPA}_\delta^\varepsilon)$. In P_0 we have $a \leftrightarroweq a + \delta$. This is no longer true if we add a TSS P_1 which contains a single axiom $x \xrightarrow{\surd} x$.

Another example of this kind is given by Rules 8 and 9 in Table 2 of Section 6.1. Consider $P(\mathrm{BPA}_\delta^\varepsilon)$ to which Rule 7 has been added. None of the τ-laws holds in this system. However, if Rules 8 and 9 are added the

τ-laws do hold. Hence, Rules 8 and 9 do not preserve conservativity up to bisimulation.

7.5.5. Example. Our last example shows that non-well-foundedness of P_0 can disturb conservativity. Suppose P_0 consists of $P(\mathrm{BPA}_\delta^\varepsilon)$ and a circular (non-well-founded) rule

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

One can easily see that $\delta \leftrightarrows_{P_0} \delta + \delta$. However, adding a TSS P_1 with a single axiom $ok \xrightarrow{ok} ok$ makes $\delta \not\leftrightarrows_{P_0 \oplus P_1} \delta + \delta$.

The next theorem shows that in some sense 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 symbol 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. We use the same type of strategy as in the proof of Theorem 5.10. 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 ordinal induction on the structure 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 *tyft* and contain no function symbols from Σ_0 in the left hand side of their conclusions, r must be in R_0. Suppose r is pure *tyft* (the case that r is pure *tyxt* 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. We then have

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

Consider the dependency graph G of the premises of r. As in the proof of Theorem 5.10 we define, for each node x of G, $\mathrm{depth}(x) \in \mathbb{N}$ as the length of the maximal backward chain of edges. Further we define

$$X = \{x_i \mid 1 \leqslant i \leqslant r(f)\}$$
$$Y = \{y_i \mid i \in I\}$$
$$Y_n = \{y \in Y \mid \text{depth}(y) = n\} \qquad \text{for} \quad n \geqslant 0.$$

With induction on n we prove that $\sigma(x)$ is in $T(\Sigma_0)$ for all $x \in X \cup Y$. Because $s \in T(\Sigma_0)$ and $\sigma(f(x_1, ..., x_{r(f)})) = s$, $\sigma(x) \in T(\Sigma_0)$ for all $x \in X$. Let $n \in \mathbb{N}$ and suppose that $\sigma(x) \in T(\Sigma_0)$ for all $x \in X \cup Y_0 \cup \cdots \cup Y_{n-1}$. Let $y^* \in Y_n$. There is a unique $i \in I$ with $y^* = y_i$. Because $y_i \in Y_n$ and rule r is pure, $\text{Var}(t_i) \subseteq X \cup Y_0 \cup \cdots \cup Y_{n-1}$. But now we can apply the induction hypothesis: since $s_i = \sigma(t_i) \in T(\Sigma_0)$, $s_i' = \sigma(y_i) \in T(\Sigma_0)$ too. Since y^* is chosen arbitrarily, $\sigma(y) \in T(\Sigma_0)$ for all $y \in Y_n$. This finishes the induction on n so that we have shown that $\sigma(y) \in T(\Sigma_0)$ for all $x \in X \cup Y$. Since $\text{Var}(t) \subseteq X \cup Y$, we may conclude that $s' = \sigma(t) \in T(\Sigma_0)$. ▌

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 of the original system on the lhs of its conclusion are quite strong. It will be difficult to strengthen Theorem 7.6 by weakening these constraints. Because modularity is an important and desirable property and because TSSs which are not pure are ill-behaved with respect to modularity, one might decide, for this reason, to call such TSSs unstructured.

The main reason we had for including Theorem 7.6 in this paper is that we need it in the next section. We expect that a lot more can be said about modular properties of TSSs than we have done here.

8. Trace Congruences

In this section we study the trace congruences 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 symbols 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 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 TSSs which can be added to it conservatively. Consequently it is also difficult to characterize the completed trace congruence induced by this format. However, for TSSs 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.

8.1. DEFINITION. Let $\mathcal{A} = (S, A, \rightarrow)$ be a LTS. A state $s \in S$ 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 $a_1 * \cdots * a_n \in A^*$ is a *completed trace* of s if there are states $s_0, ..., s_n \in S$ such that $s_0 = s$ and $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s_n \not\rightarrow$. $CT(s)$ 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 $\mathscr{F}$ be some format of TSS rules. Let $P = (\Sigma, A, R)$ be a TSS in $\mathscr{F}$ format. Two terms $s, t \in T(\Sigma)$ are *completed trace congruent with respect to* $\mathscr{F}$ rules, notation $s \equiv_{\mathscr{F}} t$, if for every TSS $P' = (\Sigma', A', R')$ in $\mathscr{F}$ 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 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. *Overview of results of Section* 8. Abramsky (1987) and Bloom, Istrail, and Meyer (1988) give 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 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 prove is best illustrated by Fig. 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. For $m \in \mathbb{N}$, $\leftrightarrows^m$ is m-nested simulation equivalence. $\sim_{\mathscr{L}_m}$ is the equivalence induced by the set $\mathscr{L}_m$ of Hennessy–Milner formulas in which no negation symbol

$$\begin{tikzcd}
 & \leftrightarrows^2 \arrow[dl, "1"'] & \leftrightarrows^m \arrow[d, "4"] & \\
\equiv_{\text{pure } tyft/tyxt} \arrow[dr, "2"'] & & & \leftrightsquigarrow^m \arrow[ul, "6 \text{ (IF)}"'] \\
 & \sim_{\mathfrak{L}_2} \arrow[uu, "3 \text{ (IF)}"] & \sim_{\mathfrak{L}_m} \arrow[ur, "5 \text{ (IF)}"'] &
\end{tikzcd}$$

FIGURE 11

occurs nested m times or more. In the right corner of Fig. 11 we have an auxiliary equivalence notion $\leftrightsquigarrow^m$. In Sections 8.5–8.7 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 prove the following Theorem 8.4.2.

8.4.1. DEFINITION. An LTS $\mathcal{A} = (S, A, \rightarrow)$ is *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*

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

Proof. Direct from Theorem 8.5.8, Corollary 8.6.7, and Corollary 8.7.6 of this section. ▮

We are quite sure that, if one uses infinitary Hennessy–Milner logic as in (Milner, 1989), the restriction of image finiteness in Theorem 8.4.2 can be dropped. Because we wanted to keep the presentation as simple as possible, we preferred to leave this generalization as an exercise to the reader.

In Section 8.8 we show that, using the results that were needed to characterize the completed trace congruence for the pure *tyft/tyxt* format, it is easy to prove that the trace congruence with respect to this format coincides with simulation equivalence for image finite processes.

Bloom, Istrail, and 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 (Bloom, 1988):

8.4.3. THEOREM (Bloom, Istrail, and 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*

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

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, and 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. *Nested Simulation Equivalences.*

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 $s \underset{\rightarrow}{\subset} t$, if there is a simulation containing the pair (s, t). s and t are *simulation equivalent*, notation $s \leftrightarrows t$, if $s \underset{\rightarrow}{\subset} t$ and $t \underset{\rightarrow}{\subset} s$.

Note the difference between simulation equivalence and bisimulation equivalence: in the case of a bisimulation equivalence, there should be a single relation which is a simulation relation in two directions; in the case of simulation equivalence it is required that there be two simulation relations, one for each direction.

8.5.2. Definition. Let $\mathcal{A} = (S, A, \rightarrow)$ be a LTS and let α be an ordinal number. We define the relation $\underset{\rightarrow}{\subset}^{\alpha} \subseteq S \times S$ inductively as follows:

> $s \underset{\rightarrow}{\subset}^{\alpha} t$ iff for each $\beta < \alpha$ there is a simulation relation $R \subseteq (\underset{\rightarrow}{\subset}^{\beta})^{-1}$ with $s\, R\, t$.

Two states s and t are *α-nested simulation equivalent*, notation $s \leftrightarrows^{\alpha} t$, if $s \underset{\rightarrow}{\subset}^{\alpha} t$ and $t \underset{\rightarrow}{\subset}^{\alpha} s$.

8.5.3. Lemma. *Let $\mathcal{A} = (S, A, \rightarrow)$ be a LTS. Let α, β be ordinal numbers with $\beta < \alpha$. Let $s, t \in S$. Then:*

0. $\underset{\rightarrow}{\subset}^{0} = S \times S$
1. $\underset{\rightarrow}{\subset}^{1} = \underset{\rightarrow}{\subset}$ *and* $\leftrightarrows^{1} = \leftrightarrows$
2. $\underset{\rightarrow}{\subset}^{\alpha} \subseteq \underset{\rightarrow}{\subset}^{\beta}$
3. $\underset{\rightarrow}{\subset}^{\alpha} \subseteq (\underset{\rightarrow}{\subset}^{\beta})^{-1}$
4. $\leftrightarrows^{\alpha} \subseteq \underset{\rightarrow}{\subset}^{\alpha} \subseteq \leftrightarrows^{\beta}$
5. $\underline{\leftrightarrow} \subseteq \leftrightarrows^{\beta}$
6. $s \underset{\rightarrow}{\subset}^{\alpha+1} t$ *iff there is a simulation relation* $R \subseteq (\leftrightarrows^{\alpha})^{-1}$ *with* $s\, R\, t$.

Proof. Straightforward using the definitions. ∎

Besides the above lemma, there are many other interesting facts about nested simulations that one may try to prove. In particular it is interesting to see what are the exact relationships between nested simulation equivalences and bisimulation equivalence. Below some results are presented which clarify these relationships. Since these results are a bit outside the scope of this paper, all proofs have been omitted.

8.5.4. COUNTEREXAMPLE. Below we present a counterexample which shows that the inclusion of Lemma 8.5.3.5 is strict. In order to present the example it is useful (although not necessary) to introduce the *summation* operator $\sum$. This operator, which for instance occurs in (Milner, 1989), does not fit the framework of this paper because it may have an arbitrary, possibly infinite number of arguments. If t_i $(i \in I)$ are terms, then $\sum_{i \in I} t_i$ is a term too. Its behaviour is described by rules (for all $a \in A$, $j \in I$)

$$\frac{t_j \xrightarrow{a} y}{\sum_{i \in I} t_i \xrightarrow{a} y}.$$

One has to assume an upper bound on the cardinality of the index set I in order to make the collection of terms setlike. In our framework the operator $\sum$ can be coded by viewing $\sum_{i \in I} t_i$ as a constant. Besides the $\sum$ operator, we use δ and $+$ as in $P(\mathrm{BPA}_\delta^\varepsilon)$ and prefixing operators a: $(\cdot)$ as in Section 5.11.3. We define the following terms:

$$s_0 = c:\delta$$

$$t_0 = s_0 + b:\delta$$

$$s_{\alpha+1} = a:t_\alpha$$

$$t_{\alpha+1} = s_{\alpha+1} + a:s_\alpha.$$

If α is a limit ordinal, then

$$S_\alpha = \sum_{\beta < \alpha} d:s_\beta$$

$$s_\alpha = \sum_{\beta < \alpha} d:(S_\alpha + d:t_\beta)$$

$$t_\alpha = s_\alpha + d:S_\alpha.$$

A part of the transition system is displayed in Fig. 12. One can prove that for every ordinal α: $s_\alpha \leftrightarrows^\alpha t_\alpha$ and $s_\alpha \not\leftrightarrow t_\alpha$. However, within a fixed transition system $\leftrightarrows^\alpha$ and $\leftrightarrow$ will coincide when α is large enough:

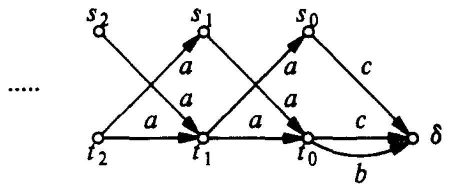


FIGURE 12

8.5.5. THEOREM. *Let $\mathcal{A} = (S, A, \rightarrow)$ be a LTS and let α be the smallest regular cardinal larger than the cardinality of all sets $\{s' \mid s \xrightarrow{a} s'\}$ $(a \in A, s \in S)$. Then*

$$s \leftrightarrows^{\alpha} t \Leftrightarrow s \leftrightarrow t.$$

This theorem implies in particular that for image finite transition systems the intersection for all $m \in \mathbb{N}$ of m-nested simulation equivalence coincides with bisimulation equivalence.

Another implication is that if, relative to some transition system, $\leftrightarrows^{\alpha}$ is different from $\leftrightarrow$, $\leftrightarrows^{\beta}$ and $\leftrightarrows^{\gamma}$ are different for all $\beta < \gamma \leqslant \alpha$.

8.5.6. *Nested Simulations and Completed Trace Equivalence.* Simulation equivalence does not refine completed trace equivalence. Take for example the simulation equivalent processes a and $a\delta + a$. The completed trace sets are $\{a * \surd\}$ and $\{a, a * \surd\}$, respectively. However, it is not hard to see that for $m \geqslant 2$, m-nested simulation equivalence does refine completed trace equivalence.

8.5.7. LEMMA. *Let $\Sigma = (F, r)$ be a signature and let $P = (\Sigma, A, R)$ be a TSS. If P is well-founded and in tyft/tyxt format, then for all ordinals α, $\leftrightarrows^{\alpha}$ is a congruence for all function symbols in F.*

Proof. Completely analogous to the proof of Theorem 5.10. Let P be well-founded and in *tyft/tyxt* format. It is sufficient to show that for all ordinals α, all $f \in F$, and all closed terms $u_i, v_i \in T(\Sigma)$ $(1 \leqslant i \leqslant r(f))$,

$$\forall i \; u_i \subsetneq^{\alpha} v_i \Rightarrow f(u_1, ..., u_{r(f)}) \subsetneq^{\alpha} f(v_1, ..., v_{r(f)}).$$

We prove this statement with induction on α. Let α be an ordinal and suppose the statement is proved for $\beta < \alpha$. Let $R \subseteq T(\Sigma) \times T(\Sigma)$ be the least relation satisfying

— $\subsetneq^{\alpha} \subseteq R$,

— for all function symbols f in F and terms u_i, v_i $(1 \leqslant i \leqslant r(f))$ in $T(\Sigma)$,

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

It is enough to show $R \subseteq \underrightarrow{\subset}^{\alpha}$. Let $\beta < \alpha$. Since, by Lemma 8.5.3.3, $\underrightarrow{\subset}^{\alpha} \subseteq (\underrightarrow{\subset}^{\beta})^{-1}$, and because, by the induction hypothesis, $\underrightarrow{\subset}^{\beta}$ is a congruence we have that $R \subseteq (\underrightarrow{\subset}^{\beta})^{-1}$. In order to show that $R \subseteq \underrightarrow{\subset}^{\alpha}$, it remains to be shown that R is a simulation relation, i.e., if $u\,R\,v$ and $u \xrightarrow{a}_P u'$, then there is a v' such that $v \xrightarrow{a}_P v'$ and $u'\,R\,v'$. The proof of this fact can in essence be copied from the proof of Theorem 5.10. ▮

The next theorem states the validity of inclusion 1.

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

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

Proof. Let $s, t \in T(\Sigma)$ with $s \underline{\leftrightarrows}^2 t$. 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, $s \underline{\leftrightarrows}^2 t$ within $TS(P \oplus P')$. Now we use that $\underline{\leftrightarrows}^2$ is a congruence for operators in pure *tyft/tyxt* format (Lemma 8.5.7) and get $C[s] \underline{\leftrightarrows}^2 C[t]$. Since $\underline{\leftrightarrows}^2$ 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. *Testing Hennessy–Milner Formulas.* Next we give the definitions of Hennessy–Milner logic (HML) and prove the second inclusion in Fig. 11. Most definitions are standard and can also be found in (Hennessy and Milner, 1985). 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 $\mathscr{M}$ of (Hennessy and Milner, 1985).

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

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

Let $\mathscr{A} = (S, A, \rightarrow)$ be an LTS. The satisfaction relation $\models \subseteq S \times \mathscr{L}$ 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 $\mathscr{L}'$ 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 $\mathcal{A} = (S, A, \rightarrow)$ be a LTS and let $\mathscr{K}$ be a set of HML formulas. With $\sim_{\mathscr{K}}$ we denote the equivalence relation on S induced by $\mathscr{K}$:

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

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

We recall a fundamental result of Hennessy and Milner (1985):

8.6.3. Theorem (Hennessy and Milner). *Let $\mathcal{A} = (S, A, \rightarrow)$ be an image finite LTS. Then for all $s, t \in S$,*

$$s \leftrightarrow t \Leftrightarrow s \sim_{\mathscr{L}} t.$$

8.6.4. Definition. For $m \in \mathbb{N}$ define the set $\mathscr{L}_m$ of HML-formulas given by:

— $\mathscr{L}_0$ is empty,

— $\mathscr{L}_{m+1}$ is from the following grammar:

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

We leave it as an exercise to the reader to check that the equivalence induced by $\mathscr{L}_m$ formulas is the same as the equivalences induced by the sets $\mathscr{K}_m^{\langle\,\rangle}$ and $\mathscr{K}_m^{[\,]}$ which are given by

— $\mathscr{K}_0^{\langle\,\rangle} = \mathscr{K}_0^{[\,]} = \varnothing$.

— $\mathscr{K}_{m+1}^{\langle\,\rangle}$ is defined by

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

— $\mathscr{K}_{m+1}^{[\,]}$ is defined by

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

8.6.5. EXAMPLE. Consider the terms s_i, t_i as defined in Section 8.5.4. Define for $0 < m < \omega$ the formula $\varphi_m \in \mathscr{L}_m$ 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 $i \geqslant 0$: $s_i \not\models \varphi_{i+1}$ and $t_i \models \varphi_{i+1}$.

8.6.6. THEOREM (Testing $\mathscr{L}_2$ 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 $\mathscr{L}_2$ 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 symbols $+$ and *Sat*. Observe that the signature is finite if the alphabet A_0 is finite. For δ and $+$ we have just the same rules as in BPA$^{\varepsilon}_{\delta}$ and a: denotes prefixing as in Example 5.11.3. The most interesting operator is the operator *Sat*. Its first argument is intended to be a coding of some $\mathscr{L}_2$ formula. The *Sat* operator tests whether its second argument satisfies the $\mathscr{L}_2$ 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 from Theorem 7.6 that P_1 is a conservative extension of P_0.

TABLE 4

A Test System for $\mathscr{L}_2$ Formulas

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

$\mathcal{L}_2$ formulas are 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 $t \models \phi$. With this claim, which we prove below, we can finish the proof of Theorem 8.6.6: whenever for some $s, t \in T(\Sigma_0 \oplus \Sigma_1)$ with $s \not\sim_{\mathcal{L}_2} t$, there is an $\mathcal{L}_2$ formula ϕ_0 such that $s \models \phi_0$ and $t \not\models \phi_0$ (or vice versa). Using the claim this means $Sat(C_{\phi_0}, s) \not\equiv_{CT} Sat(C_{\phi_0}, t)$.

Before we present a formal proof of the claim, we give some intuition about how $Sat(C_\phi, t)$ tests the formula ϕ on t. If $\phi = T$, testing is straightforward: $C_T = skip{:}\delta$ and *skip* indicates to *Sat* that it can do an *ok* step (rule 1). Hence, $Sat(skip{:}\delta, t) \xrightarrow{ok} Sat(\delta, t)$ and it is not hard to check that $Sat(\delta, t)$ cannot do a next step.

Testing of $\wedge$ ad $\langle a \rangle$ is almost as straightforward as testing the formula T and resembles the definition of $\models$. The intuitive meaning of the constant symbols *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 $ok * ok$, which is not *ok* because now $ok \notin CT(Sat(C_\phi, t))$. Next we give a formal proof of the claim.

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 $Sat(C_\phi, t) \xrightarrow{ok} Sat(\delta, t)$ and $Sat(\delta, t)$ has no outgoing transitions. 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. ∎

Claim. *Let $t \in T(\Sigma_0 \oplus \Sigma_1)$ and let $\phi \in \mathscr{L}_2$. Then*

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

Proof. ($\Rightarrow$) Induction on the structure of ϕ.

(a) ϕ is $\neg\psi$, $\psi \in \mathscr{L}_1$. We have $t \not\models \psi$. By the lemma above, $CT(Sat(C_\psi, t)) = \varnothing$. By Rule 1, $Sat(C_\phi, t) \xrightarrow{ok} Sat(C_\psi, t)$. Hence ok is 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 $\neg\psi$, $\psi \in \mathscr{L}_1$. 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 $\psi \in \mathscr{L}_1$, the lemma 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 $t \models \phi$.

This completes the proof of Theorem 8.6.6. ▮

8.6.7. Corollary (Inclusion 2). *Let P be a TSS in pure tyft/tyxt format. Then*

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

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

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

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

Proof. Suppose that $s \underrightarrow{\subset}^m t$ and $s \models \phi$ for some $\phi \in \mathscr{L}_m$. We prove $t \models \phi$ with induction on m. The case $m = 0$ is trivial. So suppose $m > 0$. We prove $t \models \phi$ with induction on the structure of ϕ.

(a) ϕ is $\neg \psi$, $\psi \in \mathscr{L}_{m-1}$. By definition of $s \underrightarrow{\subset}^m t$ we have $t \underrightarrow{\subset}^{m-1} s$. 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 $t \models \phi_2$. 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 $s \underrightarrow{\subset}^m t$, there exists an m-nested simulation R containing (s, t). Hence, for some $t' \in S$, $t \xrightarrow{a} t'$ and $s' \mathrel{R} t'$. So $s' \underrightarrow{\subset}^m t'$. By induction $t' \models \phi'$ and thus $t \models \phi$. ▮

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

8.7.2. Definition. Let $\mathfrak{A} = (S, A, \rightarrow)$ be a LTS. Define for $m, n \in \mathbb{N}$ relations $\underset{\rightsquigarrow}{\subset}^m_n \subseteq S \times S$ by

— $s \underset{\rightsquigarrow}{\subset}^m_0 t$ always,

— $s \underset{\rightsquigarrow}{\subset}^0_n t$ always,

— $s \underset{\rightsquigarrow}{\subseteq}{}^{m+1}_{n+1} t$ iff $t \underset{\rightsquigarrow}{\subseteq}{}^{m}_{n+1} s$ and whenever $s \xrightarrow{a} s'$ then there is a t' such that $t \xrightarrow{a} t'$ and $s' \underset{\rightsquigarrow}{\subseteq}{}^{m+1}_{n} t'$.

We write

— $s \underset{\rightsquigarrow}{\leftrightarrows}{}^{m}_{n} t$ if $s \underset{\rightsquigarrow}{\subseteq}{}^{m}_{n} t$ and $t \underset{\rightsquigarrow}{\subseteq}{}^{m}_{n} s$,

— $s \underset{\rightsquigarrow}{\leftrightarrows}{}^{m} t$ if for all n: $s \underset{\rightsquigarrow}{\leftrightarrows}{}^{m}_{n} t$,

— $s \underset{\rightsquigarrow}{\subseteq}{}^{m} t$ if for all n: $s \underset{\rightsquigarrow}{\subseteq}{}^{m}_{n} t$.

8.7.3. Lemma. *Let $m, n \in \mathbb{N}$. Then $\underset{\rightsquigarrow}{\subseteq}{}^{m}_{n+1} \subseteq \underset{\rightsquigarrow}{\subseteq}{}^{m}_{n}$ and $\underset{\rightsquigarrow}{\leftrightarrows}{}^{m}_{n+1} \subseteq \underset{\rightsquigarrow}{\leftrightarrows}{}^{m}_{n}$.*

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 $m \in \mathbb{N}$:*

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

Proof. Suppose that $s \not\underset{\rightsquigarrow}{\leftrightarrows}{}^{m} t$. With induction on m we show that there is a $\phi \in \mathcal{L}_m$ such that $s \models \phi$ but $t \not\models \phi$. It cannot be that $m = 0$. So take $m > 0$. Since $s \not\underset{\rightsquigarrow}{\leftrightarrows}{}^{m} t$, there must be an n such that $s \not\underset{\rightsquigarrow}{\leftrightarrows}{}^{m}_{n} t$. 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 $n > 0$. If $t \not\underset{\rightsquigarrow}{\leftrightarrows}{}^{m-1}_{n} s$ then we can find, by the induction hypothesis, 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, on the other hand, $t \underset{\rightsquigarrow}{\subseteq}{}^{m-1}_{n} s$, 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 $t \xrightarrow{a} t'$: $s' \not\underset{\rightsquigarrow}{\leftrightarrows}{}^{m}_{n-1} t'$. 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 $t_1, \ldots, t_p$ that can be reached from t by an a-transition. Since $s' \not\underset{\rightsquigarrow}{\leftrightarrows}{}^{m}_{n-1} t_i$ 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 $\phi = \phi_1 \wedge \cdots \wedge \phi_p$. 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 $m \in \mathbb{N}$,*

$$s \underset{\rightsquigarrow}{\leftrightarrows}{}^{m} t \Rightarrow s \underset{\rightarrow}{\leftrightarrows}{}^{m} t.$$

Proof. Suppose that $s \underset{\rightsquigarrow}{\subseteq}{}^{m} t$. With induction on m we prove that $s \underset{\rightarrow}{\subseteq}{}^{m} t$. The case $m = 0$ is trivial. So suppose $m > 0$. We prove that $\underset{\rightsquigarrow}{\subseteq}{}^{m}$ is an m-nested simulation relation. Whenever $v \underset{\rightsquigarrow}{\subseteq}{}^{m} w$ then for all n, $v \underset{\rightsquigarrow}{\subseteq}{}^{m}_{n} w$. Hence by definition of $\underset{\rightsquigarrow}{\subseteq}{}^{m}$, $w \underset{\rightsquigarrow}{\subseteq}{}^{m-1}_{n} v$ for all n. Thus $w \underset{\rightsquigarrow}{\subseteq}{}^{m-1} v$ and by induction $w \underset{\rightarrow}{\subseteq}{}^{m-1} v$. So the relation $\underset{\rightsquigarrow}{\subseteq}{}^{m}$ is contained in the relation $(\underset{\rightarrow}{\subseteq}{}^{m-1})^{-1}$. It remains to be shown that $\underset{\rightsquigarrow}{\subseteq}{}^{m}$ is a simulation relation.

Suppose $v \underset{\rightsquigarrow}{\subseteq}^m w$ and $v \xrightarrow{a} v'$. Since for all $n > 0$, $v \underset{\rightsquigarrow}{\subseteq}^m_n w$ there is for each n a w_n such that $w \xrightarrow{a} w_n$ and $v' \underset{\rightsquigarrow}{\subseteq}^m_{n-1} w_n$. Due to the image finiteness there must be a w^* that occurs infinitely often in the sequence $w_1, w_2, \ldots$. Because for all n $\underset{\rightsquigarrow}{\subseteq}^m_{n-1} \supseteq \underset{\rightsquigarrow}{\subseteq}^m_n$ by Lemma 8.7.3, we have that for all $n > 0$, $v \underset{\rightsquigarrow}{\subseteq}^m_{n-1} w^*$ and therefore $v \underset{\rightsquigarrow}{\subseteq}^m w^*$. This concludes the proof that $\underset{\rightsquigarrow}{\subseteq}^m$ 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 $m \in \mathbb{N}$,*

$$s \leftrightarrows^m t \Leftrightarrow s \underset{\rightsquigarrow}{\leftrightarrows}^m t \Leftrightarrow s \sim_{\mathcal{L}_m} t.$$

Proof. Immediate from Theorems 8.7.1, 8.7.4, and 8.7.5. ∎

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 LTSs). We just repeat the argumentation above for trace congruence instead of completed trace congruence. First the notion of trace congruence is defined.

8.8.1. DEFINITION. Let $\mathcal{A} = (S, A, \rightarrow)$ be a LTS. A sequence $a_1 * \cdots * a_n \in A^*$ is a *trace* of s if there are states $s_1, s_2, \ldots, s_n \in S$ such that $s \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s_n$. $T(s)$ 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 $s \equiv_T t$.

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, notation $s \equiv^T_{\mathcal{F}} t$, 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 $C[\;]$, $C[s] \equiv_T C[t]$.

8.8.3. 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*

$$s \equiv^T_{pure\ tyft/tyxt} t \Leftrightarrow s \leftrightarrows t \Leftrightarrow s \sim_{\mathcal{L}_1} t.$$

Proof. In fact most of the work has already been done. The equivalence of $\leftrightarrows$ and $\sim_{\mathcal{L}_1}$ follows from Corollary 8.7.6. The implication $s \equiv^T_{\text{pure}\ tyft/tyxt} t \Rightarrow s \leftrightarrows t$ follows by Lemma 8.5.7 and the observation that simulation equivalence refines trace equivalence. The reverse implication 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 8.4.3 for tree rules of Bloom, Istrail, and Meyer follows from

Theorem 8.5.8, 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, and 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 $\mathscr{L}_2$ 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 chose 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 $\mathscr{L}_2$ 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 synchronization of actions in a set $H \subseteq A_1$. For these operators R_1 contains rules ($a \in A_1$):

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

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

Next Σ_1 contains a binary operator *Sat* which sets whether its second argument satisfies the $\mathscr{L}_2$ 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 $a \in A_1$). If A_0 is finite then clearly A_1, Σ_1, and R_1 are finite too. Let the mapping $s\colon \mathscr{L}_1 \to \mathbb{N}$ 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

$$S_0 = ok : \delta$$

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

TABLE 5

A Test System for $\mathscr{L}_2$ Formulas with Tree Rules Only

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

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

$$\begin{aligned} C_T &= ok:\delta \\ C_{\phi \wedge \psi} &= left:C_\phi + right:C_\psi \\ C_{\neg\phi} &= size:S_{s(\phi)} + neg:C_\phi \\ C_{\langle a \rangle \phi} &= \langle\ \rangle:a:C_\phi. \end{aligned}$$

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

CLAIM. *Let $\phi \in \mathscr{L}_2$ 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 $\phi \in \mathscr{L}_1$. This is a direct consequence of the next lemma which can be proved easily by means of induction on the structure of ϕ:

LEMMA 1. *Let $\phi \in \mathscr{L}_1$ with $s(\phi) = n$ and let $t \in T(\Sigma_0 \oplus \Sigma_1)$. Then*:

— $t \models \phi \Rightarrow \{i^n * ok\} \subseteq CT(Sat(C_\phi, t)) \subseteq \{i^n * ok\} \cup \{i^m \mid 1 \leqslant m \leqslant n\}$,

— $t \not\models \phi \Rightarrow CT(Sat(C_\phi, t)) \subseteq \{i^m \mid 1 \leqslant m \leqslant n\}$.

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 a *skip-i* operator around $Sat(C_\phi, t)$ in a structured way $s(\phi)$ times 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 instructions on how 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) an *ok* step will always be generated, whereas a subsequent *ko* action is generated only when the tested process satisfies ϕ. One can prove the following lemma:

LEMMA 2. *Let* $\phi \in \mathscr{L}_1$ *with* $s(\phi) = n$ *and let* $t \in T(\Sigma_0 \oplus \Sigma_1)$. *Then*:

— $t \models \phi \Rightarrow CT(Context(S_n, Sat(C_\phi, t))) = \{i^n * ok * ko\}$,

— $t \not\models \phi \Rightarrow CT(Context(S_n, Sat(C_\phi, t))) = \{i^n * ok\}$.

Using Lemma 2, the claim can be proved with straightforward induction on the structure of ϕ. Theorem 8.9.1 is an immediate consequence of the claim. ▮

9. Comparison with Other Formats

In this section we will give an extensive comparison of our format with the formats proposed by de Simone (1984, 1985) and Bloom, Istrail, and Meyer (1988). First both formats are described.

9.1. DEFINITION. Let $\Sigma = (F, r)$ be a signature and let A be a set of labels. A *De Simone rule* (over Σ and A) takes the form

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

where:

— $f \in F$ and $r(f) = l$,

— $I \subseteq \{1, ..., l\}$,

— $x_1, ..., x_l$ and y_i $(i \in I)$ are distinct variables

(for $1 \leqslant i \leqslant l$ let $x_i' = y_i$ if $i \in I$ and $x_i' = x_i$ otherwise),

— t is a term in $T(\Sigma, \{x_1', ..., x_l'\})$ in which each x_i' occurs at most once.

Clearly the de Simone 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. 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

$$\frac{\{x_i \xrightarrow{a_{ij}} y_{ij} \mid 1 \leqslant i \leqslant l, 1 \leqslant j \leqslant m_i\} \cup \{x_i \stackrel{b_{ij}}{\not\rightarrow} \mid 1 \leqslant i \leqslant l, 1 \leqslant j \leqslant n_i\}}{f(x_1, ..., x_l) \xrightarrow{a} t},$$

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 triple (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 by Bloom, Istrail, and Meyer (1988). There a GSOS rule system contains some additional ingredients for dealing with guarded recursion and there are a number of finiteness constraints. The feature with distinguishes GSOS rules from the other rules in this paper is the possibility of *negative* premises. This makes it 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 $\rightarrow \subseteq T(\Sigma) \times A \times T(\Sigma)$ *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 are 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 Bloom, Istrail, and Meyer (1988) shows that in general 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:

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

$$\frac{x \overset{a}{\nrightarrow}}{g(x) \xrightarrow{a} d}$$

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

There is no transition relation which agrees with these rules. In particular, $f(c)$ can move iff it cannot move.[1]

9.3. Examples. Below we list some examples that illustrate the differences between the formats.

9.3.1. *Global Closure Properties.* Rules in *tyxt* format fit neither de Simone's format nor 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.

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 de Simone or 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 $\mathscr{L}_2$ formulas, this type of rules play an important role. In (Baeten and van Glabbeek, 1987), operators ε_K are described that erase all actions from a set $K \subseteq Act$. We can add these operators to $P(\text{BPA}_\delta^\varepsilon)$ together with the following rules from (Baeten and van Glabbeek, 1987):

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

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

The same type of trick can also be used to describe the "atomic version operator." This operator was introduced by de Bakker and Kok (1988) for giving semantics to concurrent Prolog. Here we give our own variant of this operator, using our own notation. The interested reader who wants to

[1] In (Groote, 1989), it is investigated in which cases a specification in *ntyft*/*ntyxt* format is consistent. A general method, based on the stratification technique in logic programming, is presented to show consistency of sets of rules. It is shown that various results from this paper extend smoothly to a setting where rules may contain negative premises.

know how this type of operator can be used to give semantics to concurrent Prolog is referred to (de Bakker and Kok, 1988). Take as starting point the signature of $\mathrm{BPA}_\delta^\varepsilon$. But as labels of transitions we now take not elements of $Act_{\surd}$, but elements of the set of finite sequences over $Act_{\surd}$. 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 $R(\mathrm{BPA}_\delta^\varepsilon)$ (but now the labels should be interpreted as sequences!) and moreover the following rules:

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

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

The rules express that only successful sequences, i.e., sequences ending on $\surd$, can happen in the scope of an atomic version operator. The rules are in *tyft* format. Hence, strong bisimulation is a congruence in this setting.

9.3.3. *Lookahead.* All operators defined with the de Simone 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 van Glabbeek and Vaandrager (1987), cannot be defined:

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

Other examples of operators with a 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 (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 $P(\mathrm{BPA}_{\varepsilon\delta}^\tau)$ as described in Section 6.1, then we can derive

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

Observe that the *rules* that contain a function symbol τ_I all have a lookahead of 1 (i.e., the length of the maximal path in the dependency graphs of the rules is 1). As *operators* on transition systems the τ_I have an unbounded lookahead, due to the presence of *tyxt* rules with a lookahead of 2 in $P(\mathrm{BPA}^{\tau}_{\varepsilon\delta})$.

9.3.4. *Copying.* In contrast to de Simone's format, the GSOS format and also our format can describe operators which copy their arguments. The system call *fork* of Unix (1986) is a typical example of an operation that one would like to describe using copying. One can think of a rule such as

$$fork(x) \xrightarrow{\tau} parent(x) \parallel child(x).$$

Below we present another example where copying 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 symbol 0, a unary function symbol *succ*, and binary function symbols + and ×. There is only one transition label, namely 1. The operational semantics of the operators is described by the following rules:

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

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

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

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

9.3.5. *Branching.* The ability to copy arguments is not the only difference between de Simone's format and the GSOS format. A rule such as

$$\frac{x \xrightarrow{a} x',\, x \xrightarrow{b} x''}{f(x) \xrightarrow{a} f(x')}$$

fits the GSOS format but not de Simone's format. In this rule we see a *branching* in the dependency graph at node x.

9.3.6. *Catalysis.* A similar example is obtained if we add to $P(\mathrm{BPA}^{\varepsilon}_{\delta})$ the following rule which fits the GSOS 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's format, 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.

9.3.7. *Priorities.* In (Baeten, Bergstra, and Klop, 1986) 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 (Baeten, Bergstra, and Klop, 1986) the operator is defined using equations, but if one uses Plotkin-style rules then it is inevitable to use negative hypotheses.

Consider the GSOS rule system $P(\mathrm{BPA}_\delta^\varepsilon)$ 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

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

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. Cleaveland and Hennessy (1988) describe priorities using *tyxt* rules with negative hypotheses. Another example of an operator that is defined using rules with negative premises is the *broadcast* operator as described by Pnueli (1985).

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 $\mathscr{L}_2$ 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 or the *tyft/tyxt* format. Now it is a standard result that the completed trace congruence induced by languages such as Meije-SCCS, ACP, and CSP coincides with *failure equivalence* ($\equiv_\mathrm{F}$) (see for instance (Bergstra, Klop, and Olderog, 1988)). Hence the completed trace congruence induced by de Simone's format is failure equivalence (it is not too difficult to give a direct proof of this fact).

Bloom, Istrail, and Meyer (1988) characterized the completed trace congruence induced by their format in terms of the equivalence corresponding to the following set of formulas:[2]

[2] The formulas as defined in (Bloom, Istrail, and Meyer, 1988) were called *limited modal formulas* and may also contain F and $\vee$. However, it is easily proved that this addition does not increase their distinguishing power.

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

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

9.4.2. Theorem (Bloom, Istrail, and Meyer, 1988). *Let $P = (\Sigma, A, R)$ be a GSOS rule system such that the associated transition system is image finite. Then $\equiv_{GSOS} = \sim_{\mathscr{D}}$.*

Some additional insight is provided by the following characterization of denial equivalence which is due to Larsen and Skou (1989).

9.4.3. Definition. Let $\mathcal{A} = (S, A, \rightarrow)$ be a LTS. A relation $R \subseteq S \times S$ is a *$\frac{2}{3}$-bisimulation*, also called a *ready simulation*, 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 *$\frac{2}{3}$-bisimilar* (or *ready simulation equivalent*) in $\mathcal{A}$ if there exist a $\frac{2}{3}$-bisimulation containing the pair (s, t) and a $\frac{2}{3}$-bisimulation containing the pair (t, s).

9.4.4. Theorem (Larsen and Skou, 1989). *Let $\mathcal{A} = (S, A, \rightarrow)$ be an image finite LTS. Then two states are $\frac{2}{3}$-bisimular just in case they satisfy exactly the same denial formulas.*

It is a trivial exercise to show that

$$\leftrightarrows^2 \subseteq \underline{\leftrightarrow}_{2/3} \subseteq \equiv_F \subseteq \equiv_{CT}.$$

The examples of Figs. 13, 14, and 15 show that these inclusions are strict.

9.4.5. *Testing Denial Formulas.* The question arises whether all features of the GSOS format are really needed to test denial formulas. In

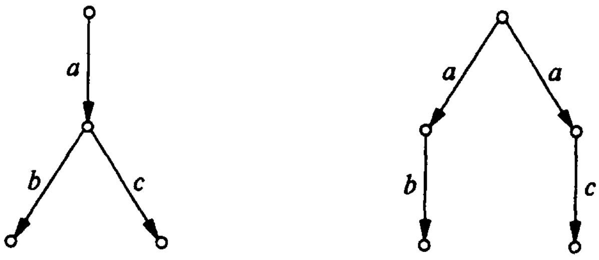


Fig. 13. Completed trace equivalent but not de Simone congruent.

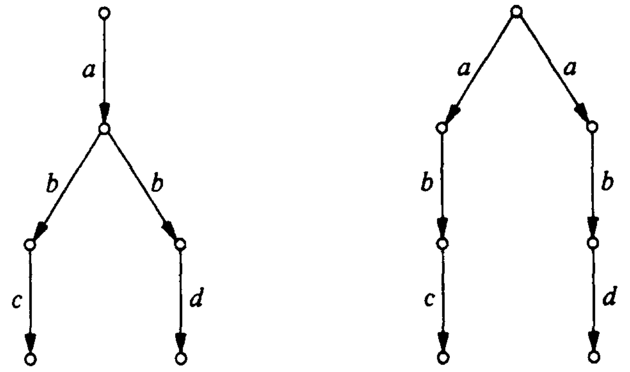


FIG. 14. de Simone congruent but not GSOS congruent.

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 van Glabbeek (1988), 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 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 nonbranching, 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.*

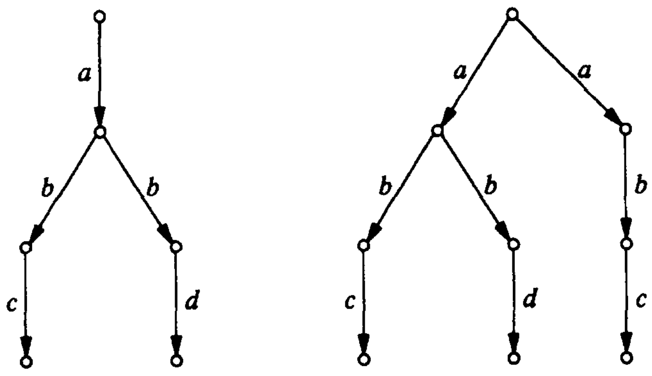


FIG. 15. GSOS congruent but not pure *tyft/tyxt* congruent.

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 symbols $+$, $\|$, *Sat*, $Sat_{[\ ]}$, $Sat_{\langle\ \rangle}$, and Sat_{right}. The rules for δ, a:, and $+$ 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. One can check that P_1 can be added conservatively to P_0.

Denial formulas are encoded using the following rules:

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

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

TABLE 6

A Test System for Denial Formulas

$\dfrac{x \xrightarrow{[\ ]} x'}{Sat(x,y) \xrightarrow{[\ ]} Sat_{[\ ]}(x',y)}$	$\dfrac{x \xrightarrow{a} x' \quad y \xrightarrow{a} y'}{Sat_{[\ ]}(x,y) \xrightarrow{ko} \delta}$
$\dfrac{x \xrightarrow{\langle\rangle} x'}{Sat(x,y) \xrightarrow{\langle\rangle} Sat_{\langle\rangle}(x',y)}$	$\dfrac{x \xrightarrow{a} x' \quad y \xrightarrow{a} y'}{Sat_{\langle\rangle}(x,y) \xrightarrow{ok} Sat(x',y')}$
$\dfrac{x \xrightarrow{left} x'}{Sat(x,y) \xrightarrow{left} Sat(x',y) \| Sat_{right}(x,y)}$	$\dfrac{x \xrightarrow{right} x'}{Sat_{right}(x,y) \xrightarrow{right} Sat(x',y)}$
$\dfrac{x \xrightarrow{a} x'}{x \| y \xrightarrow{a} x' \| y}$	$\dfrac{y \xrightarrow{a} y'}{x \| y \xrightarrow{a} x \| y'}$

(see for instance (De Nicola and Hennessy, 1984; Bergstra, Klop, and Olderog, 1988)): these languages allow one to observe *traces* and *deadlock* and to *block* actions from a certain moment onwards. This makes it possible to detect *refusals* indirectly: one concludes that a certain action can be refused after an initial trace because deadlock occurs if all the other actions are blocked. 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 the de Simone format is the capacity to make copies of processes at every moment. Observe that the only rule in Table 6 that does not fit 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 $\mathscr{L}_2$ 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 de Simone's format that a certain action is refused because deadlock occurs when the other actions are blocked, one can conclude in the *tyft/tyxt* format 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 (1981) calls *controlling the weather conditions*. Abramsky (1987) 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 able to see negative information (such as the absence of some action) directly, and because it is also not able to force all nondeterministic branches to be 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 and Skou (1989) 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 arbitrarily 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 and Skou (1989) 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, $\mathscr{L}_2$ 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, and Meyer (1988) argue that bisimulation equivalence *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, and Meyer (1988). And indeed what is 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 (Bloom, Istrail, and Meyer, 1988), are motivated by the requirement that the transition relation should be computably finitely branching. We think that, although it is certainly important 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, Istrail, and Meyer (1988) seem to do). 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 TSSs right from the start, then one misses a large number of interesting applications. Of course the question of what type of TSSs do lead to computably finitely branching transition systems is a very interesting one. It seems that if one generalizes the positive GSOS format in the direction of the *tyft*/*tyxt* format, infinite branching arises quite soon. The example in Fig. 16, for instance, which is due to Bard Bloom, illustrates that function symbols in the premises are "dangerous."

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 Fig. 16 shows that ω has an infinite number of outgoing transitions. Another example illustrating the same point is obtained by adding recursion to $P(\mathrm{BPA}_\delta^\varepsilon)$ 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 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

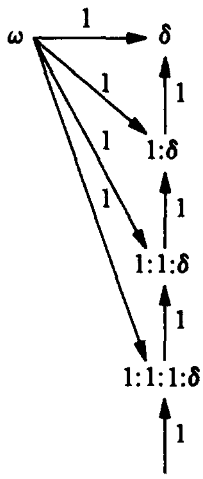


FIGURE 16

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.

ACKNOWLEDGMENTS

We 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. Finally, we thank the referee of this paper.

RECEIVED February 27, 1989; FINAL MANUSCRIPT RECEIVED November 20, 1990

REFERENCES

ABRAMSKY, S. (1987), Observation equivalence as a testing equivalence, *Theoret. Comput. Sci.* **53**, 225–241.

AMERICA, P., DE BAKKER, J. W., KOK, J. N., AND RUTTEN, J. J. M. M. (1986), Operational semantics of a parallel object-oriented language, *in* "onference Record of the 13th ACM Symposium on Principles of Programming Languages (POPL), St. Petersburg, Florida," pp. 194–208.

BAETEN, J. C. M., AND BERGSTRA, J. A. (1988), Global renaming operators in concrete process algebra, *Inform. and Comput.* **78**, 205–245.

BAETEN, J. C. M., BERGSTRA, J. A., AND KLOP, J. W. (1986), Syntax and defining equations for an interrupt mechanism in process algebra, *Fund. Inform.* **9**, No. 2, 127–168.

BAETEN, J. C. M., AND VAN GLABBEEK, R. J. (1987), Merge and termination in process algebra, *in* "Proceedings, Conference on Foundations of Software Technology and Theoretical Computer Science, Pune, India" (K. V. Nori, Ed.), pp. 153–172, Lecture Notes in Computer Science, Vol. 287, Springer-Verlag, Berlin/New York.

DE BAKKER, J. W., AND KOK, J. N. (1988), Uniform abstraction, atomicity and contractions in the comparative semantics of concurrent Prolog, *in* "Proceedings, Fifth Generation Computer Systems 1988 (FGCS 88), Tokyo, Japan," pp. 347–355.

BERGSTRA, J. A., AND KLOP, J. W. (1988), A complete inference system for regular processes with silent moves, *in* "Proceedings, Logic Colloquium 1986" (F. R. Drake and J. K. Truss, Eds.), pp. 21–8–, North-Holland, Hull; also appeared as Report CS-R8420, Centrum voor Wiskunde en Informatica, Amsterdam, 1984.

BERGSTRA, J. A., KLOP, J. W., AND OLDEROG, E.-R. (1988), Readies and failures in the algebra of communicating processes, *SIAM J. Comput.* **17**, No. 6, 1134–1177.

BLOOM, B. (November 1988), personal communication.

BLOOM, B., ISTRAIL, S., AND MEYER, A. R. (1988), Bisimulation can't be traced: Preliminary report, *in* "Conference Record of the 15th ACM Symposium on Principles of Programming Languages (POPL), San Diego, California," pp. 229–239. Full version appeared as Technical report TR 90-1150, Cornell University, Ithaca, New York, August 1990.

BOUDOL, G. (1985), Notes on algebraic calculi of processes, *in* "Logics and Models of Concurrent Systems" (K. Apt, Ed.), pp. 261–303, NATO ASI Series F13, Springer-Verlag, Berlin/New York.

CLEAVELAND, R., AND HENNESSY, M. (1988), Priorities in process algebra, *in* "Proceedings, 3rd Annual Symposium on Logic in Computer Science (LICS), Edinburgh," pp. 193–202.

DE NICOLA, R., AND HENNESSY, M. (1984), Testing equivalences for processes, *Theoret. Comput. Sci.* **34**, 83–133.

van Glabbeek, R. J. (1987), Bounded nondeterminism and the approximation induction principle in process algebra, *in* "Proceedings STACS 87" (F. J. Brandenburg, G. Vidal-Naquet, and M. Wirsing, Eds.), pp. 336–347, Lecture Notes in Computer Science, Vol. 247, Springer-Verlag, Berlin/New York.

van Glabbeek, R. J. (November 1988), personal communication.

van Glabbeek, R. J., and Vaandrager, F. W. (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, and P. C. Treleaven, Eds.), pp. 224–242, Lecture Notes in Computer Science, Vol. 259, Springer-Verlag, Berlin/New York.

Groote, J. F. (1989), "Transition System Specifications with Negative Premises," Report CS-R8950, Centrum voor Wiskunde en Informatica, Amsterdam; extended abstract *in* "Proceedings, CONCUR 90, Amsterdam" (J. C. M. Baeten and J. W. Klop, Eds.), pp. 332–341, Lecture Notes in Computer Science, Vol. 458, Springer-Verlag, Berlin/New York.

Hennessy, M. (1988), "Algebraic Theory of Processes," MIT Press, Cambridge, MA.

Hennessy, M., and Milner, R. (1985), Algebraic laws for nondeterminism and concurrency, *J. Assoc. Comput. Mach.* **32**(1), 137–161.

Keller, R. M. (1976), Formal verification of parallel programs, *Comm. ACM* **19**(7), 371–384.

Klop, J. W. (1987), Term rewriting systems: A tutorial, *Bull. European Assoc. Theoret. Comput. Sci.* **32**, 143–182.

Larsen, K. G., and Skou, A. (1989), Bisimulation through probabilistic testing, *in* "Conference Record of the 16th ACM Symposium on Principles of Programming Languages (POPL), Austin, Texas," pp. 344–352.

Milner, R. (1980), "A Calculus of Communicating Systems," Lecture Notes in Computer Science, Vol. 92, Springer-Verlag, Berlin/New York.

Milner, R. (1981), Modal characterization of observable machine behaviour, *in* "Proceedings CAAP 81" (G. Astesiano and C. Bohm, Eds.), pp. 25–34, Lecture Notes in Computer Science, Vol. 112, Springer-Verlag, Berlin/New York.

Milner, R. (1983), Calculi for synchrony and asynchrony, *Theoret. Comput. Sci.* **25**, 267–310.

Milner, R. (1989), Communication and Concurrency, Prentice-Hall, Englewood Cliffs, NJ.

Olderog, E.-R., and Hoare, C. A. R. (1986), Specification-oriented semantics for communicating processes, *Acta Informat.* **23**, 9–66.

Park, D. M. R. (1981), Concurrency and automata on infinite sequences, *in* "Proceedings, 5th GI Conference" (P. Deussen, Ed.), pp. 167–183, Lecture Notes in Computer Science, Vol. 104, Springer-Verlag, Berlin/New York.

Plotkin, G. D. (1981), "A Structural Approach to Operational Semantics," Technical Report DAIMI FN-19, Computer Science Department, Aarhus University.

Plotkin, G. D. (1983), An operational semantics for CSP, *in* "Proceedings IFIP TC2 Working Conference on Formal Description of Programming Concepts—II, Garmisch, 1982" (D. Bjørner, Ed.), pp. 199–225, North-Holland, Amsterdam.

Pnueli, A. (1985), Linear and branching structures in the semantics and logics of reactive systems, *in* "Proceedings ICALP 85, Nafplion" (W. Brauer, Ed.), pp. 15–32, Lecture Notes in Computer Science, Vol. 194, Springer-Verlag, Berlin/New York.

de Simone, R. (1984), "Calculabilité et expressivité dans l'algèbre de processus parallèles Meije, Thèse de 3^e cycle, Univ. Paris 7.

de Simone, R. (1985), Higher-level synchronising devices in Meije-SCCS, *Theoret. Comput. Sci.* **37**, 245–267.

UNIX (1986), "Programmer's Reference Manual, 4.3 BSD Edition," Computer Systems Research Group, University of California, Berkeley.

Vrancken, J. L. M. (1986), "The Algebra of Communicating Processes with Empty Process," Report FVI 86-01, Dept. of Computer Science, University of Amsterdam.