Structured Operational Semantics and Bisimulation as a Congruence

(extended abstract)

Jan Friso Groote
Frits Vaandrager
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands
Email: jfg@cwi.nl & fritsv@cwi.nl

In this paper the question is considered in which cases a transition system specification in Plotkin style has 'good' properties and deserves the predicate 'structured'. The discussion takes place in a setting of labelled transition systems. The states of the transition systems are terms generated by a single sorted signature and the transitions between states are defined by conditional rules. We argue that in this setting it is natural to require that strong bisimulation equivalence is a congruence on the states of the transition systems. A general format, called the *tyft/tyxt* format, is presented for the conditional rules in a transition system specification, such that bisimulation is always a congruence when all the rules fit into this format. With a series of examples it is demonstrated that the *tyft/tyxt* format cannot be generalized in any obvious way. Briefly we touch upon the issue of modularity of transition system specifications. We show that certain pathological *tyft/tyxt* rules (the ones which are not *pure*) can be disqualified because they behave badly with respect to modularisation. Next we address the issue of full abstraction. We characterize the completed trace congruence induced by the operators in pure *tyft/tyxt* format as *2-nested simulation equivalence*. The pure *tyft/tyxt* format includes the format given by DE SIMONE [16,17] but is incomparable to the GSOS format of BLOOM, ISTRAIL & MEYER [7]. However, it turns out that 2-nested simulation equivalence strictly refines the completed trace congruence induced by the GSOS format.

1. INTRODUCTION

In [14, 15] PLOTKIN advocates a simple method for giving operational semantics to programming languages. The method, which is often referred to as *SOS* (for *Structured Operational Semantics*), is based on transition systems. The states of the transition systems are terms in some formal language that, in general, will extend the language for which one wants to give a semantics. The main idea of the method is that the transitions between states are defined by conditional rules.

Nowadays Plotkin's method has become rather popular and a large number of (concurrent) languages have been provided with an operational semantics in SOS style. Therefore it might be worthwhile to consider in more detail the questions how expressive different classes of transition system specifications (TSS's) are and in which cases a TSS has good properties.

The following desirable properties of TSS's are stated by BLOOM, ISTRAIL & MEYER [7], as requirements to be fulfilled by any reasonably structured TSS.

1. existence of a canonical system of transition relations agreeing with the rules,
2. availability of structural induction as a proof technique,
3. the TSS leads to transition systems which are computably finitely branching,
4. strong bisimulation is a congruence.

Let us consider these requirements in more detail.

(1) The first requirement clearly makes sense but will not be much of a problem for us, since in

1. The research of the authors was supported by ESPRIT project no. 432, An Integrated Formal Approach to Industrial Software Development (METEOR), and by RACE project no. 1046, Specification and Programming Environment for Communication Software (SPECS). A full version of this paper appeared as [9]. There also the proofs can be found which have been omitted here.

this paper we consider only Plotkin style conditional rules with *positive* hypothesis. In this case the initial algebra approach guarantees the existence of a natural transition relation: a transition is there iff it has a proof. BLOOM, ISTRAIL & MEYER [7] also consider rules with *negative* premises. In this case the first requirement becomes less trivial.
(2) Since the title of Plotkin's original paper ([14]) is 'A structural approach to operational semantics', one may argue that the first S in SOS should stand for 'structural' rather than 'structured'. Apparently, Plotkin used the word 'structural' because of its connection with structural induction on abstract syntax. However, by now there are many examples of interesting TSS's, which are commonly accepted as specifications in the SOS style, but which contain rules which clearly are not compatible with structural induction. Besides the standard example of the rule for recursion, other examples are described for instance in [2-4, 8]. The point is that one can appeal to more general induction principles. In this paper we will mostly use induction on the structure of the proofs of transitions.
(3) We think that, although it is certainly pleasant to have finiteness and decidability, it is much too strong to call any TSS leading to a transition relation which does not have these properties 'not reasonably structured'. If one disqualifies infinitary and undecidable TSS's right from the start, then one misses a large number of interesting applications. We will describe a rule format that gives us the expressiveness to describe the invisible nature of τ (see section 3.11). Therefore it is to be expected that, in general, we also have the infinite branching and undecidability of the models of CCS/ACP_τ based on observation equivalence.
(4) A fundamental equivalence on the states of a labelled transition system is the strong bisimulation equivalence of PARK [13]. Strong bisimulation equivalence seems to be the *finest* extensional behavioural equivalence one would want to impose, i.e. two states of a transition system which are bisimilar cannot be distinguished by external observation. This means that from an observational point of view, the transition systems generated by the SOS approach are too concrete as semantical objects. The objects that really interest us will be *abstract* transition systems where the states are bisimulation equivalence classes of terms, or maybe something even more abstract. If bisimulation is not a congruence then this means that the function that returns the transitions associated to a phrase when given the transitions associated to its immediate components, depends on properties of the transition system which are generally considered to be irrelevant, such as the specific names of states. Hence we think that a transition system specification which leads to transition systems for which bisimulation is not a congruence should not be called structured: possibly it is compositional on the level of (concrete) transition systems but it is not compositional on the more fundamental level of transition systems modulo bisimulation equivalence.

Summarizing, we agree with BLOOM, ISTAIL & MEYER [7] that requirements 1 and 4 are essential, but we think that their requirements 2 and 3 are too strong in general. 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 (positive) TSS, 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 signatures and substitution. We define the notion of a transition system specification (TSS) and describe how a TSS determines a transition system. Moreover the fundamental notion of strong bisimulation is introduced.

In section 3 we present a general format, called the *tyft/tyxt* format, for the rules in a TSS and prove that bisimulation is always a congruence when all rules have this format (and a small additional technical condition is satisfied). With a series of examples it is demonstrated that this format cannot be generalized in any obvious way. Section 3 also 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 [2, 12], and elsewhere. If the rules in a TSS do not fit in our format then there is a good chance that something will be wrong: either

bisimulation is not a congruence right away or the congruence property will get lost if more operators and rules are added.

A natural and important operation on transition system specifications P_0,P_1 is to take their componentwise union $P_0 \oplus P_1$. A desirable property 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 4 we show that most of the *tyft/tyxt* rules (the rules which are *pure*) behave fine under modularisation. Rules that are not pure behave badly under modularisation, but fortunately these rules are quite pathological.

Central in the theory of concurrency is the idea that processes which cannot be distinguished by observation, should be identified: a process semantics should be *fully abstract* with respect to some notion of testing. Mostly one takes the position that the observations one can make on a process include its *completed traces*, i.e. the (finite) maximal sequences of actions which can be performed by a process. Two processes are *completed trace congruent* with respect to some format of rules if they yield the same completed traces in any context that can be built from operations defined in this format. The main result of section 5 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 [10] with no [] in the scope of a $\langle\rangle$. Consequently the following two trees, which are not bisimilar, cannot be distinguished by operators defined with pure *tyft/tyxt* rules:

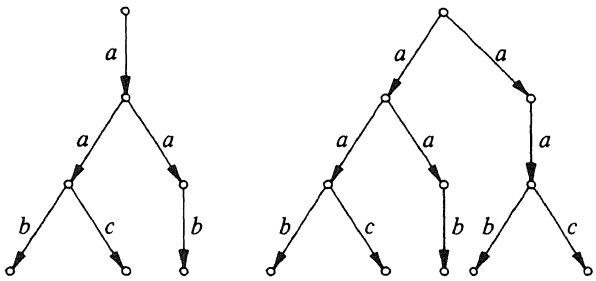

FIGURE 1. Pure *tyft/tyxt* congruent but not bisimilar

Many process equivalences can be based on some notion of testing, a framework of extracting information about a system by doing experiments on it. ABRAMSKY [1], for instance, develops a notion of testing for bisimulation equivalence which incorporates a hierarchy of increasingly powerful testing constructs: traces, refusals, copying and global testing. In the full version of this paper, we adress the question whether there exists a reasonable notion of testing for 2-nested simulation equivalence. *tyft/tyxt* languages allow one to observe *traces* and to detect *refusals* indirectly: one concludes that a certain action is refused because some completed trace is not there. In addition it is allowed to make *copies* of processes at every moment. Finally, the *lookahead* in the *tyft/tyxt* rules makes it possible to investigate *all* branches of a process for positive information and to see whether a certain *tree* is possible. Because the lookahead does not allow one to see negative information (like the absence of some action) directly, and because it is also not able to force that all nondeterministic branches are pursued by some number of copies, lookahead does not give one the full testing power of global testing. Bloom, Istrail & Meyer argue that, unlike copying, global testing is not realistic. We think that, unless one believes in fortune telling as a technique which has some practical relevance for computer science, also lookahead as a testing notion is not very realistic. Still, lookahead pops up naturally if one looks at the maximal format of rules for which bisimulation is a congruence and we

argued that rules with a lookahead are often useful. Therefore we think that, just like bisimulation equivalence, 2-nested simulation equivalence is interesting and worth studying.

The full version of this paper contains an extensive comparison of our format with the format proposed by DE SIMONE [16, 17] and the GSOS format of BLOOM, ISTRAIL & MEYER [7]. Roughly speaking, 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 [7] 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 into the GSOS format but which contain no negation. We call the corresponding format *positive GSOS*. BLOOM, ISTRAIL & MEYER [7] 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 []. This implies that 2-nested simulation equivalence refines GSOS trace congruence. In [9], we show that the completed trace congruence induced by the *positive* GSOS format equals the GSOS trace congruence. So although the general GSOS format can be used to define certain operations which cannot be defined using positive rules only, the use of negations in the definition of operators does not introduce any new distinctions between processes.

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

2. BASIC DEFINITIONS

Throughout this paper we assume the presence of a countably infinite set V of *variables* with typical elements $x,y,z...$

2.1. DEFINITION. A *(single sorted) signature* Σ is a pair (F,r) where F is a set of *function names* disjoint with V, and $r:F\to\mathbb{N}$ is a *rank function* which gives the arity of a function name; if $f\in F$ and $r(f)=0$ then f is called a *constant name*. With $\mathbb{T}(\Sigma)$, we denote the set of open terms over signature Σ (so these terms may contain variables from V). $T(\Sigma)$ denotes the set of closed or ground terms over Σ. $Var(t)\subseteq V$ denotes the set of variables in a term t. A *substitution* σ is a mapping in $V\to\mathbb{T}(\Sigma)$. It is extended to a mapping $\sigma:\mathbb{T}(\Sigma)\to\mathbb{T}(\Sigma)$ in the standard way. If σ and ρ are substitutions, then substitution $\sigma\circ\rho$ is defined by: $\sigma\circ\rho(x) = \sigma(\rho(x))$ for $x\in V$.

2.2. 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 a finite 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 satisfying the above format, then the elements of $\{t_i \xrightarrow{a_i} t_i' \mid i\in I\}$ are called the *premises* of r and $t\xrightarrow{a}t'$ is called the *conclusion* of r. A rule of the form $\frac{\emptyset}{t\xrightarrow{a}t'}$ 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 letters $\phi,\psi,\chi,..$ will be used to range over transitions. The notions 'substitution', '*Var*' and 'closed' extend to transitions and rules as expected.

2.3. DEFINITION. Let $P=(\Sigma,A,R)$ be a TSS. A *proof* of a transition ψ from P is a finite, upwardly branching tree of which the nodes are labelled by transitions $t \xrightarrow{a} t'$ with $t,t' \in \mathbb{T}(\Sigma)$ and $a \in A$, such that: the root is labelled with ψ, and 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 $\dfrac{\{\phi_i \mid i \in I\}}{\phi}$ in R and a substitution σ 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.

2.4. 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.*

As a running example we present below a TSS for a simple process language.

2.5. EXAMPLE. Let $Act=\{a,b,c,..\}$ be a given set of *actions*. We consider the signature $\Sigma(\mathrm{BPA}_\delta^\epsilon)$ (Basic Process Algebra with δ and ϵ) of [18]. $\Sigma(\mathrm{BPA}_\delta^\epsilon)$ contains constants a for each $a \in Act$, a constant δ that stands for *deadlock*, and a constant ϵ that denotes the *empty process*, a process that terminates immediately and successfully. 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.
The TSS $P(\mathrm{BPA}_\delta^\epsilon)$ consists of signature $\Sigma(\mathrm{BPA}_\delta^\epsilon)$, labels $Act_\surd$, and the rules of table 1. There a ranges over $Act_\surd$, unless further restrictions are made. Infix notation is used for the binary function names.

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

TABLE 1

An operational semantics makes use of some sort of (abstract) machines and describes how these machines behave. Here we take as machines simply nondeterministic automata in the sense of classical automata theory, also called labelled transition systems.

2.6. DEFINITION. A *(nondeterministic) automaton* or *labelled transition system (LTS)* is a structure $(S,A,\longrightarrow)$ where S is a set of *states*, A is an *alphabet*, and $\longrightarrow \subseteq S \times A \times S$ is a *transition relation*. Elements $(s,a,s') \in \longrightarrow$ are called *transitions* and will be written as $s \xrightarrow{a} s'$.

The notion of strong bisimulation equivalence as defined below is from PARK [13].

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

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

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

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

Here the relation $\rightarrow_P \subseteq T(\Sigma)\times A\times T(\Sigma)$ is defined by: $t\xrightarrow{a}_P t' \Leftrightarrow P\vdash t\xrightarrow{a}t'$. We say that two terms $t,t'\in T(\Sigma)$ are (*P-*)*bisimilar*, notation $t\leftrightarrows_P t'$, if $TS(P):t\leftrightarrows t'$. We write $t\leftrightarrows t'$ if it is clear from the context what P is. Note that $\leftrightarrows_P$ is also an equivalence relation.

2.9. EXAMPLE. For the TSS $P(\mathrm{BPA}_\delta^\epsilon)$ of example 2.5 one can derive identities (a)-(e) below. In (f) it is shown that the left distributivity of $\cdot$ over $+$ does not hold in bisimulation semantics. Like in regular algebra we will often omit the $\cdot$ in a product $x\cdot y$ and we take $\cdot$ to be more binding than $+$. Missing brackets in expressions xyz and $x+y+z$ associate to the right.

(a)	$\epsilon\epsilon \leftrightarrows \epsilon$	(d)	$b\epsilon \leftrightarrows b$
(b)	$b \leftrightarrows b+b$	(e)	$\epsilon b \leftrightarrows b$
(c)	$(\epsilon a+\epsilon b)(cd\delta+\delta) \leftrightarrows (a(c+\delta)d+bc(d+d))\delta$	(f)	$ab+ac \not\leftrightarrows a(b+c)$

3. COMPOSITIONAL TRANSITION SYSTEM SPECIFICATIONS

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

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

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

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

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

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

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

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 $P(\mathrm{BPA}_\delta^\epsilon)$ of example 2.5 is in *tyft/tyxt* format, but also any TSS obtained from this TSS by dropping some arbitrary rules.

3.2. Circularity. A TSS with the rule:

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

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

3.2.1. DEFINITION. Let $P=(\Sigma,A,R)$ be a TSS. Let $S=\{t_i \xrightarrow{a_i} t_i' \mid i\in I\}$ be a set of transitions of P. The *dependency graph* of S is a directed (unlabelled) graph with:

- Nodes: $\bigcup_{i\in I} Var(t_i \xrightarrow{a_i} t_i')$,
- Edges: $\{<x,y> \mid x\in Var(t_i), y\in Var(t_i') \text{ for some } i\in I\}$.

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

3.2.2. EXAMPLE. The dependency graph of the rule in section 3.2 is given in figure 2. The rule is circular since the graph clearly contains a cycle.

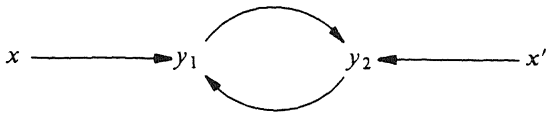

FIGURE 2

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

Hence, two TSS's are equivalent if they have the same signature, the same set of labels and if the sets of rules determine the same transition relation. The particular form of the rules is not important. In example 2.5 for instance, we can replace rule 6 of table 1 by the rule:

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

The resulting TSS $P'(\mathrm{BPA}_\delta^\varepsilon)$ is equivalent to $P(\mathrm{BPA}_\delta^\varepsilon)$. The reason for this is that whenever $P(\mathrm{BPA}_\delta^\varepsilon)$ proves a transition of the form $t \xrightarrow{\surd} t'$, t' will be syntactically equal to δ. Observe that $P'(\mathrm{BPA}_\delta^\varepsilon)$ is not in *tyft/tyxt* format. We will come back to this in section 3.10.

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

3.5. DEFINITION. Let $P=(\Sigma,A,R)$ be a TSS and let r be a rule in R. A variable in $Var(r)$ is *free* if it does not occur in the left hand side of the conclusion or in the right hand side of a premise. Rule r is *pure* if it is non circular and contains no free variables. TSS P is *pure* if all its rules are pure.

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

We now come to the first main theorem of this paper.

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

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

3.8. COUNTEREXAMPLES. Before we commence with the proof of this theorem, we present a number of examples which show that the condition in the theorem that the TSS is in *tyft/tyxt* format cannot be weakened in any obvious way. At present, we have no example to show that the condition that the TSS is non circular cannot be missed: we just have not been able to prove the theorem without it. However, circular TSS's are so exotic that we doubt whether they will ever be used. In section 4 it will be shown that circular rules are ill-behaved with respect to modularisation.

The first example shows that in general the variables in the left side of the arrow in the conclusion must all be different. It is obtained by adding to $P(\mathrm{BPA}_\delta^\epsilon)$ the axiom $x+x \xrightarrow{ok} \delta$. We then have $a \leftrightarrows a\epsilon$, but $a+a \not\leftrightarrows a+a\epsilon$ as a and $a\epsilon$ are not syntactically equal.

In general there may not appear more than one function name at the left of the transition predicate in the conclusion. Take the TSS $P(\mathrm{BPA}_\delta^\epsilon)$ extended with the axiom $x+(y+z) \xrightarrow{ok} \delta$. We have $b \leftrightarrows b+b$, but $b+(b+b) \not\leftrightarrows b+b$.

Our next example shows that in the premises the right hand side of a transition are not allowed to contain function names. We add *prefixing* operators $a{:}(\cdot)$ to $P(\mathrm{BPA}_\delta^\epsilon)$ for each $a \in Act$ and define the operational meaning of these operators with axioms $a{:}x \xrightarrow{a} x$. If we now add moreover the rule:

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

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

The variables at the right hand side of the arrows in the premises must in general be different. This is shown by adding the rule:

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

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

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

3.9. Proof of theorem 3.7. Let $\Sigma=(F,r)$ be a signature and let $P=(\Sigma,A,R_0)$ be a non circular TSS in *tyft/tyxt* format. We have to prove that $\leftrightarrows_P$ is a congruence. Let $R \subseteq T(\Sigma)\times T(\Sigma)$ be the least relation satisfying:

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

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

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

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

Lemma 2.4 says that there is a proof for $f(u_1,..,u_{r(f)}) \xrightarrow{a} u'$ that only contains closed

transitions. We will prove the statement with induction on the structure of this proof. Lemma 3.6 allows us to assume throughout the proof that the rules in R_0 are pure and in *tyft* format.

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

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

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

FACT. *Let* $t \in \mathbb{T}(\Sigma)$ *and let* $\sigma, \sigma': V \to T(\Sigma)$ *be substitutions such that for all* x *in* $Var(t)$: $\sigma(x) R \sigma'(x)$. *Then* $\sigma(t) R \sigma'(t)$.
PROOF. Straightforward induction on the structure of t using the definition of R. □

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

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

It follows that: 1) $\sigma(x_i) = u_i$ for $1 \leqslant i \leqslant r(f)$, and 2) $\sigma(t) = u'$.
Our aim is to use the rule r again in the proof of $f(v_1,..,v_{r(f)}) \xrightarrow{a} v'$ for some v' by finding a proper substitution σ'. Define:

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

Here $X = \{x_i \mid 1 \leqslant i \leqslant r(f)\}$ and $Y = \{y_i \mid i \in I\}$. Stepwise we will extend the definition of σ' to elements of Y in such a way that for all variables x in V: $\sigma'(x)$ is defined $\Rightarrow \sigma(x) R \sigma'(x)$. Consider the dependency graph G of the premises of r. Call a node of G *coloured* if σ' is defined for this node (cq. variable). So initially we are in a state where all nodes are coloured except for the ones in Y. We will colour these nodes one by one. In the process the invariant will be preserved that whenever a node is coloured, all its predecessors are coloured too.
A term or transition is called coloured if all the variables contained in it are coloured. Hence, initially none of the transitions in $\{t_i \xrightarrow{a} y_i \mid i \in I\}$ is coloured. In the process of colouring variables we also preserve the invariant that whenever a transition $\psi \in \{t_i \xrightarrow{a} y_i \mid i \in I\}$ is coloured, i.e. σ' is defined for all variables of ψ, there exists a proof from P of $\sigma'(\psi)$.
Now we first observe that with a complete colouring that satisfies the invariant properties, the induction step can easily be finished. Due to the last invariant property there is a proof for $\sigma'(\chi_i)$ for all premises χ_i of r. Now construct a new proof with as root $\sigma'(f(x_1,..,x_{r(f)}) \xrightarrow{a} t)$ and as direct subgraphs the proofs of the $\sigma'(\chi_i)$. Define $v' = \sigma'(t)$. Clearly, we have a proof for $f(v_1,..,v_{r(f)}) \xrightarrow{a} v'$. We may also conclude that for all $x \in Var(t)$: $\sigma(x) R \sigma'(x)$. By an application of the previously proved fact it follows that $\sigma(t) R \sigma'(t)$ or equivalently $u' R v'$. This completes the induction step except for the proof that a complete colouring exists.
In order to do this, it is sufficient to show that whenever we have a colouring which satisfies the invariant properties and in which only some nodes in Y are not coloured, we can extend this colouring with one element, while preserving the invariant. Let X' be such a colouring. We claim that there is some $i \in I$ such that t_i is coloured but y_i not. In order to see that this is true, assume that there is no such i. It cannot be that for some $j \in I$, y_j is coloured, but t_j is not

coloured, because that would contradict with the assumption that all predecessors of a coloured node are coloured, too. Hence, we can partition I in two sets. $I_c = \{j \mid t_j$ and y_j are coloured$\}$ and $I_{nc} = \{j \mid t_j$ and y_j are not coloured$\}$. By assumption I_{nc} is non-empty. In the dependency graph G each element y_i with i in I_{nc} has an incoming edge from some element y_j with j in I_{nc}. Hence, G contains a cycle and we have a contradiction. So let $i \in I$ with t_i coloured but y_i not coloured. We have that P proves the transition $\sigma(t_i) \xrightarrow{a_i} \sigma(y_i)$ with a proof of depth less than n and furthermore $\sigma(t_i) R\, \sigma'(t_i)$ because for all variables $x \in Var(t_i)$ $\sigma(x) R\, \sigma'(x)$ (use the previously proved fact). By definition of R we can distinguish between two cases:

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

$$\sigma(t_j) = g(w_1,..,w_{r(g)}),$$
$$\sigma'(t_j) = g(w_1',..,w_{r(g)}') \text{ and}$$
$$w_j R\, w_j' \text{ for } 1 \leqslant j \leqslant r(g).$$

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

This completes the proof of the induction step. □

3.10. The implication in theorem 3.7 cannot be reversed. The TSS $P'(\mathrm{BPA}_\delta^\varepsilon)$ described following definition 3.3 is not in *tyft/tyxt* format. But, as observed in that section, it is equivalent to the TSS $P(\mathrm{BPA}_\delta^\varepsilon)$ which *is* in *tyft/tyxt* format. Hence, bisimulation equivalence is a congruence. However, if one adds new operators and rules, then the congruence property can get lost, even if the rules for the new operators are in *tyft/tyxt* format. 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 $a \leftrightarrows \partial_{\{b\}}(a)$, but $a \cdot b \not\leftrightarrows \partial_{\{b\}}(a) \cdot b$.

This example shows that there is another reason for using TSS's in *tyft/tyxt* format, namely their extensibility, without endangering congruence properties. It seems that, whenever a TSS contains a non *tyft/tyxt* rule, we can extend this TSS (except for some trivial cases) with a number of *tyft/tyxt* rules such that for the resulting TSS bisimulation is not a congruence.

We conclude this section with two examples of applications of our congruence theorem.

3.11. The silent move. In process algebra it is current practice to have a constant τ representing an internal machine step that cannot be observed. In order to describe the 'invisible' nature of τ, the notions of *observational congruence* [11] and *rooted-τ-bisimulation* [5] have been introduced. As observed by VAN GLABBEEK [8] it is not necessary to introduce a new notion of bisimulation. Below the signature $\Sigma(\mathrm{BPA}_\delta^\varepsilon)$ is enlarged with a constant name τ and rules are given that capture the notion of hidden, internal machine steps. $P(\mathrm{BPA}_{\varepsilon\delta}^\tau) = (\Sigma(\mathrm{BPA}_{\varepsilon\delta}^\tau), Act_{\tau\surd}, R(\mathrm{BPA}_{\varepsilon\delta}^\tau))$ with $Act_{\tau\surd} = Act_\surd \cup \{\tau\}$. $R(\mathrm{BPA}_{\varepsilon\delta}^\tau)$ consists of the combination of the rules in table 1 (but now a ranges over $Act_{\tau\surd}$) and table 2 (where a also ranges over $Act_{\tau\surd}$). We claim that the theory $\mathrm{BPA}_{\varepsilon\delta}^\tau$ of table 3 (where a ranges over elements from Act_τ), is a sound and complete axiomatisation of the model generated by the TSS $P(\mathrm{BPA}_{\varepsilon\delta}^\tau)$ modulo strong (!) bisimulation. Thus bisimulation becomes a congruence in a natural way. For more information we refer to [8].

$a \xrightarrow{a} \tau \quad (a \neq \surd)$	$\dfrac{x \xrightarrow{a} y \quad y \xrightarrow{\tau} z}{x \xrightarrow{a} z}$	$\dfrac{x \xrightarrow{\tau} y \quad y \xrightarrow{a} z}{x \xrightarrow{a} z}$

TABLE 2

$\mathrm{BPA}^{\tau}_{\epsilon\delta}$

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

TABLE 3

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

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

for every declaration $X \Leftarrow t_X$. Now observe that these rules fit in the *tyft* format. Hence it follows that bisimulation remains a congruence.

4. Modular properties of transition system specifications

Given two TSS's P_0 and P_1 we use $P_0 \oplus P_1$ to denote their componentwise union. 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.

4.1. 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:

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

4.2. Definition. Let $P_i=(\Sigma_i,A_i,R_i)$ $(i=0,1)$ be two TSS's with $\Sigma_0 \oplus \Sigma_1$ defined. The *sum* of P_0 and P_1, notation $P_0 \oplus P_1$, is the TSS $(\Sigma_0 \oplus \Sigma_1, A_0 \cup A_1, R_0 \cup R_1)$.

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

Note that the implication $P \vdash s \xrightarrow{a} t \Leftarrow P_0 \vdash s \xrightarrow{a} t$ holds trivially. Observe further that if P is a conservative extension of P_0, P is also a conservative extension of P_0 up to bisimulation, i.e. for $s,t \in T(\Sigma_0)$: $s \leftrightarrows_P t \Leftrightarrow s \leftrightarrows_{P_0} t$.

4.4. 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 no rule of R_1 contains a function name from Σ_0 in the left hand side of its conclusion. Let $P=P_0\oplus P_1$ be defined. Then P_1 can be added conservatively to P_0.*

In the full version of this paper a number of simple and easy to find counterexamples are included which show that there is no obvious generalisation of the above theorem. So TSS's which are not pure are ill-behaved with respect to modularisation. Because modularity is an important and desirable property one might decide, for this reason, to call such TSS's unstructured.

5. COMPLETED TRACE CONGRUENCE

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

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

and make that syntactically different terms always have outgoing transitions with different labels. Completed trace congruence would then just be syntactic equality between terms.
The results of the previous section show that for a TSS in *tyft/tyxt* format it is in general rather difficult to determine a class of TSS's which can be added to it conservatively. Consequently it is also difficult to characterize the completed trace congruence induced by this format. However, for TSS's in pure *tyft/tyxt* format such a class exists: by theorem 4.4 every TSS in *tyft* format can be added conservatively to a TSS in pure *tyft/tyxt* format. For this reason we decided to study the completed trace congruence induced by the pure *tyft/tyxt* format and leave the general *tyft/tyxt* format for what it is. We think that this is not a serious restriction.

5.1. DEFINITION. Let $\mathcal{A}=(S,A,\rightarrow)$ be a LTS. $s\in S$ 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 $a_1*..*a_n\in A^*$ is a *completed trace* of s if there are states $s_1,s_2,..,s_n\in S$ such that $s\xrightarrow{a_1}s_1\xrightarrow{a_2}..\xrightarrow{a_n}s_n\not\rightarrow$. $CT(s)$ is the set of all completed traces of s. Two states $s,t\in S$ are *completed trace equivalent* if $CT(s)=CT(t)$. This is denoted as $s\equiv_{CT}t$.

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

ABRAMSKY [1] and BLOOM, ISTRAIL & MEYER [7] give Plotkin style rules to define operators with which one can distinguish between any pair of non-bisimilar processes. This is not possible with pure *tyft/tyxt* rules because, as we will see, the notion of completed trace congruence with respect to pure *tyft/tyxt* rules exactly coincides with *2-nested simulation equivalence* (for *image finite* processes). This last equivalence is coarser than bisimulation equivalence.

5.3. First we define the notion of m-nested simulation equivalence for arbitrary $m \geqslant 0$.

5.3.1. DEFINITION. Let $\mathcal{A}=(S,A,\rightarrow)$ be a LTS. A relation $R \subseteq S \times S$ is a *simulation* if, whenever $s\,R\,t$ and $s \xrightarrow{a} s'$, there exists a $t' \in S$, with $t \xrightarrow{a} t'$ and $s'\,R\,t'$. *s can be simulated by t*, notation $s \leftrightarrows t$, if there is a simulation containing the pair (s,t). s and t are *simulation equivalent*, notation $s \leftrightarrows t$, if $s \leftrightarrows t$ and $t \leftrightarrows s$.

5.3.2. DEFINITION. Let $\mathcal{A}=(S,A,\rightarrow)$ be a LTS. The relations $\leftrightarrows^m$ $(m \geqslant 0)$ are defined inductively by:

i) $\leftrightarrows^0 = S \times S$,

ii) A relation $R \subseteq S \times S$ is an *$m+1$-nested simulation* if it is a simulation contained in $(\leftrightarrows^m)^{-1}$. State *$s$ can be simulated $m+1$-nested by* state t, notation $s \leftrightarrows^{m+1} t$, if there exists an $m+1$-nested simulation containing the pair (s,t).

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

Observe that 1-nested simulation equivalence is the same as simulation equivalence. Further note that for $m \geqslant 0$, $\leftrightarrows \subseteq \leftrightarrows^{m+1} \subseteq \leftrightarrows^{m+1} \subseteq \leftrightarrows^m$.

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

$$s_0 = c\delta \qquad t_0 = c\delta + b\delta$$
$$s_{m+1} = at_m \qquad t_{m+1} = as_m + at_m$$

Below in figure 3 a part of the transition system is displayed (some ϵ's are dropped). One can easily prove that: $s_m \leftrightarrows^m t_m$ and $s_m \not\leftrightarrows^{m+1} t_m$ for $m \geqslant 0$.

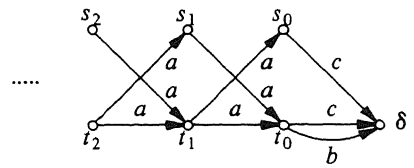

FIGURE 3

5.3.4. LEMMA. *Let $\Sigma=(F,r)$ be a signature and let $P=(\Sigma,A,R)$ be a TSS. If P is non circular and in tyft/tyxt format, then $\leftrightarrows^m$ $(m \geqslant 0)$ is a congruence for all function names in F.*

PROOF. Completely analogous to the proof of theorem 3.7. □

It is well known that simulation equivalence does not refine completed trace equivalence. Take for example the simulation equivalent processes a and $a\delta + a$. The sets of completed traces are $\{a \star \surd\}$ and $\{a, a \star \surd\}$, respectively. However, it is not hard to see that for $m \geqslant 2$, m-nested simulation equivalence does refine completed trace equivalence. This observation, together with theorem 4.4 and lemma 5.3.4 gives the following theorem.

5.3.5. THEOREM. *Let $P=(\Sigma,A,R)$ be a TSS in pure tyft/tyxt format. Then $\leftrightarrows^2 \subseteq \equiv_{\text{pure tyft/tyxt}}$.*

5.4. Hennessy-Milner logic. Next we recall the definition of Hennessy-Milner logic (HML). The definition is standard and can also be found in [10].

5.4.1. DEFINITION. The set $\mathcal{L}$ 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 $\mathcal{A}=(S,A,\rightarrow)$ be a LTS. The satisfaction relation $\models \subseteq S \times \mathcal{L}$ is defined in the standard way ($s \models \langle a\rangle\phi$ iff for some $t \in S$: $s \xrightarrow{a} t$ and $t \models \phi$).

We write F for $\neg T$, $\phi \vee \psi$ for $\neg(\neg\phi \wedge \neg\psi)$, and $[a]\phi$ 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 $\mathcal{L}'$ which is generated by the following grammar:

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

5.4.2. DEFINITION. Let $(S,A,\rightarrow)$ be a LTS and let $\mathcal{K}$ be a set of HML formulas. With $\sim_{\mathcal{K}}$ we denote the equivalence relation on S induced by $\mathcal{K}$: $s \sim_{\mathcal{K}} t \Leftrightarrow (\forall \phi \in \mathcal{K}: s \models \phi \Leftrightarrow t \models \phi)$. We will call this relation $\mathcal{K}$*-formula equivalence.*

5.4.3. DEFINITION. An LTS $(S,A,\rightarrow)$ is *image finite* if for all $s \in S$ and $a \in A$ the set $\{t \mid s \xrightarrow{a} t\}$ is finite.

We recall a fundamental result of HENNESSY & MILNER [10]:

5.4.4. THEOREM. *Let $(S,A,\rightarrow)$ be an image finite LTS. Then for all $s,t \in S$: $s \leftrightarrows t \Leftrightarrow s \sim_{\mathcal{L}} t$.*

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

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

We leave it as an exercise to the reader to check that the equivalence induced by $\mathcal{L}_m$ formulas is the same as the one induced by the set of $\mathcal{L}'$ formulas with at most n alternations of $\langle\rangle$ and $[]$.

5.4.6. EXAMPLE. Consider example 5.3.3 Define for $m \geq 1$ the formula $\varphi_m \in \mathcal{L}_m$ by: $\varphi_1 = \langle b\rangle T \wedge \langle c\rangle T$ and $\varphi_{m+1} = \langle a\rangle\neg\varphi_m$. It is easily checked that for $i \geq 0$: $s_i \not\models \varphi_{i+1}$ and $t_i \models \varphi_{i+1}$.

5.4.7. THEOREM *(Testing $\mathcal{L}_2$ formulas). Let $P_0=(\Sigma_0,A_0,R_0)$ be a TSS in pure tyft/tyxt format. Then there is a TSS $P_1=(\Sigma_1,A_1,R_1)$ in pure tyft format, which can be added conservatively to P_0, such that completed trace congruence within $P_0 \oplus P_1$ is included in $\mathcal{L}_2$ formula equivalence.*
PROOF (sketch). Set A_1 consists A_0, possibly extended with some additional labels to guarantee that A_1 contains at least 5 elements. Pick 5 elements from A_1 and give them the names *ok*, *left*, *right*, *syn* and *skip*. Signature Σ_1 contains a constant δ, unary function names *a:* for each $a \in A_1$, and binary function names $+$ and *Sat*. Note that Σ_1 is finite if A_0 is finite. For δ and $+$ we have the same rules as in BPA_δ and *a:* denotes prefixing like in the example of section 3.8. The most interesting operator is the operator *Sat*. The *Sat* operator tests whether its second argument satisfies the $\mathcal{L}_2$ formula which is represented by its first argument. The rules of the *Sat* operator are given in table 4. In the table *a* ranges over A_1. Because P_1 is in *tyft* format, $\Sigma_0 \cap \Sigma_1 = \varnothing$ and P_0 is pure *tyft/tyxt*, it follows with theorem 4.4 that P_1 is a conservative extension of P_0. $\mathcal{L}_2$ formulas are encoded using the following rules:

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

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 we can finish the proof: whenever for some $s,t \in T(\Sigma_0 \oplus \Sigma_1)$ with $s \not\sim_{\mathcal{L}_2} t$, then there is an $\mathcal{L}_2$ formula ϕ_0

$$\frac{x \xrightarrow{skip} x'}{Sat(x,y) \xrightarrow{ok} Sat(x',y)} \quad 1$$

$$\frac{x \xrightarrow{left} x_l,\ Sat(x_l,y) \xrightarrow{ok} y_l \qquad x \xrightarrow{right} x_r,\ Sat(x_r,y) \xrightarrow{ok} y_r}{Sat(x,y) \xrightarrow{ok} y_l + y_r} \quad 2$$

$$\frac{x \xrightarrow{syn} x',\ x' \xrightarrow{a} x'' \qquad y \xrightarrow{a} y',\ Sat(x'',y') \xrightarrow{ok} y''}{Sat(x,y) \xrightarrow{ok} y''} \quad 3$$

TABLE 4

such that $s \models \phi_0$ and $t \not\models \phi_0$ (or vice versa). This means that $Sat(C_{\phi_0},s) \not\equiv_{CT} Sat(C_{\phi_0},t)$. □

Now the next corollary is immediate:

5.4.8. COROLLARY. *Let P be a TSS in pure tyft/tyxt format. Then: $\equiv_{pure\ tyft/tyxt} \subseteq \sim_{\mathcal{L}_2}$.*

The following theorem is a variant of theorem 5.4.4. The proof is a bit more involved.

5.5. THEOREM. *Let $(S,A,\rightarrow)$ be an image finite LTS. Then for all $s,t \in S$ and $m \in \mathbb{N}$: $s \leftrightarrows^m t \Leftrightarrow s \sim_{\mathcal{L}_m} t$.*

Combination of theorem 5.3.5, corollary 5.4.8 and theorem 5.5 now leads to the following characterisation of the completed trace congruence induced by the pure *tyft/tyxt* format.

5.5.1. THEOREM. *Let $P=(\Sigma,A,R)$ be a TSS in pure tyft/tyxt format such that $TS(P)$ is image finite. Let $s,t \in T(\Sigma)$. Then: $s \equiv_{pure\ tyft/tyxt} t \Leftrightarrow s \leftrightarrows^2 t \Leftrightarrow s \sim_{\mathcal{L}_2} t$.*

5.6. Characterization theorem for tree rules. Bloom, Istrail & Meyer have studied the completed trace congruence induced by *tree rules*. Tree rules differ from pure *tyft/tyxt* rules in that they may only have variables in the premises and there may not be a single variable in the left hand side of a conclusion. Hence, one could also call this type of rules 'pure *xyft* rules'. They proved the following theorem [6]:

5.6.1. THEOREM (BLOOM, ISTRAIL & MEYER). *Let $P=(\Sigma,A,R)$ be a TSS in tree rule format such that $TS(P)$ is image finite. Let $s,t \in T(\Sigma)$. Then: $s \equiv_{tree\ rules} t \Leftrightarrow s \sim_{\mathcal{L}_2} t$.*

The result of Bloom, Istrail & Meyer follows from theorem 5.3.5, theorem 5.5 and the following theorem 5.6.2. In fact this combination gives a result which is even stronger than their result 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. The next theorem also strengthens theorem 5.4.7 because now only tree rules are used.

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

REFERENCES

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

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

[3] J.C.M. BAETEN & F.W. VAANDRAGER (1989): *An algebra for process creation*. Report CS-R89.., Centrum voor Wiskunde en Informatica, Amsterdam, to appear.

[4] J.W. DE BAKKER & J.N. KOK (1988): *Uniform abstraction, atomicity and contractions in the comparative semantics of concurrent Prolog*. In: Proceedings Fifth Generation Computer Systems 1988 (FGCS'88), Tokyo, Japan.

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

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

[7] B. BLOOM, S. ISTRAIL & A.R. MEYER (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.

[8] R.J. VAN GLABBEEK (1987): *Bounded nondeterminism and the approximation induction principle in process algebra*. In: Proceedings STACS 87 (F.J. Brandenburg, G. Vidal-Naquet & M. Wirsing, eds.), LNCS 247, Springer-Verlag, pp. 336-347.

[9] J.F. GROOTE & F.W. VAANDRAGER (1988): *Structured operational semantics and bisimulation as a congruence*. Report CS-R8845, Centrum voor Wiskunde en Informatica, Amsterdam.

[10] M. HENNESSY & R. MILNER (1985): *Algebraic laws for nondeterminism and concurrency*. JACM 32(1), pp. 137-161.

[11] R. MILNER (1980): *A Calculus of Communicating Systems*, LNCS 92, Springer-Verlag.

[12] R. MILNER (1983): *Calculi for synchrony and asynchrony*. Theoretical Computer Science 25, pp. 267-310.

[13] D.M.R. PARK (1981): *Concurrency and automata on infinite sequences*. In: Proceedings 5th GI Conference (P. Deussen, ed.), LNCS 104, Springer-Verlag, pp. 167-183.

[14] G.D. PLOTKIN (1981): *A Structural approach to operational semantics*. Technical Report DAIMI FN-19, Computer Science Department, Aarhus University.

[15] G.D. PLOTKIN (1983): *An operational semantics for CSP*. In: Proceedings IFIP TC2 Working Conference on Formal Description of Programming Concepts - II, Garmisch, 1982 (D. Bjørner, ed.), North-Holland, Amsterdam, pp. 199-225.

[16] R. DE SIMONE (1984): *Calculabilité et expressivité dans l'algebra de processus parallèles Meije*. Thèse de 3e cycle, Univ. Paris 7.

[17] R. DE SIMONE (1985): *Higher-level synchronising devices in* MEIJE-SCCS. Theoretical Computer Science 37, pp. 245-267.

[18] J.L.M. VRANCKEN (1986): *The algebra of communicating processes with empty process*. Report FVI 86-01, Dept. of Computer Science, University of Amsterdam.