The Meaning of Negative Premises in Transition System Specifications

(Extended Abstract)

Roland Bol & Jan Friso Groote

Centre for Mathematics and Computer Science

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

Email: bol@cwi.nl, jfg@cwi.nl

Abstract

We present a general theory for the use of negative premises in the rules of Transition System Specifications (TSS's). We formulate a criterion that should be satisfied by a TSS in order to be meaningful, i.e. to unequivocally define a transition relation. We also provide powerful techniques for proving that a TSS satisfies this criterion, meanwhile constructing this transition relation. Both the criterion and the techniques originate from logic programming [8, 7] to which TSS's are close.
As in [10], we show that the bisimulation relation induced by a TSS is a congruence, provided that it is in *ntyft/ntyxt*-format and can be proved meaningful using our techniques. As a running example, we study the combined addition of priorities and abstraction to Basic Process Algebra (BPA). Under some reasonable conditions we show that this TSS is indeed meaningful, which could not be shown by other methods [2, 10]. Finally, we provide a sound and complete axiomatization for this example. We have omitted most proofs here; they can be found in [3].

The first author is partly supported by the European Communities under ESPRIT Basic Research Action 3020 (Integration). The second author is supported by the European Communities under RACE project no. 1046 (SPECS) and ESPRIT Basic Research Action 3006 (CONCUR).

1 Introduction

Since its introduction in [14] Plotkin-style operational semantics has become a popular means for giving meaning to process calculi, specification and programming languages in terms of transition systems. A transition system consists mainly of a transition relation which is specified by a set of rules forming a *Transition System Specification* (*TSS*) [11]. Recently, the use of negative premises in these rules has become popular [2, 10, 4, 15], because it allows one to define more operators in an easy way. However, the logical meaning of those negative premises is not always clear. Therefore, the formal foundation of some of these articles is somewhat questionable.

In this article we provide a way to treat negative premises in general and we study some of the consequences of this treatment. The fundamental problem of negative premises in TSS's is that they cannot be proved in the same way positive premises can. In order to overcome this problem, we resort to a non-classical treatment of negation, similar to default logic [17] and logic programming [8]. Following [8], we define the *stable* transition relations of a TSS. A TSS can have any number of stable transition relations. We propose to define the semantics of a TSS P as the unique stable transition relation of P, if one exists.

In general it is difficult to show that a TSS has a unique stable transition relation. However, some techniques have been developed for this. The first technique, called *stratification*, is presented in [10]. Stratification is an intuitively appealing technique, and quite easy to use, but it is not always strong enough. Here, we introduce a more powerful technique, based on *well-founded*

models in logic programming [7]. This technique, which we call *reduction*, is more powerful than stratification, but also more difficult to use. The two techniques can be amalgamated, using reduction when necessary and stratification when possible. This is demonstrated on our running example, showing that under some reasonable conditions a transition relation can be associated with it.

Having defined the meaning of a TSS as its associated transition relation and shown how to arrive at this transition relation, we switch to the study of properties of transition relations as consequences of properties of their defining TSS's.

Throughout the paper we use an example to illustrate these techniques: a TSS specifying the operational semantics of Basic Process Algebra (BPA) extended with priorities [1] and abstraction [9, 12].

Acknowledgements. We thank Krzysztof Apt, Jos Baeten, Jan Bergstra, Alban Ponse, Fer-Jan de Vries and the referees for their valuable comments.

2 Preliminaries

In this section we provide the basic concepts of this paper: *transition relations* and *Transition System Specifications* (*TSS's*). We also present the running example.

We assume the presence of an infinite set V of *variables* with typical elements $x, y, z....$

Definition 2.1. A *(single sorted) signature* is a structure $\Sigma = (F, rank)$ where F is a set of *function names* disjoint with V and $rank : F \to \mathbb{N}$ is a function that gives the arity of a function name. We denote the set of terms over a signature Σ and a set of variables W by $T(\Sigma, W)$. $T(\Sigma, \emptyset)$ is abbreviated by $T(\Sigma)$; elements from $T(\Sigma)$ are called *closed* or *ground terms*. $\mathbb{T}(\Sigma)$ is used to abbreviate $T(\Sigma, V)$, the set of *open terms*. $Var(t) \subseteq V$ is the set of variables in a term $t \in \mathbb{T}(\Sigma)$. 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. A substitution is *closed* (or *ground*) if it maps all variables onto closed terms.

A transition relation prescribes what activities, represented by labeled transitions, can be performed by terms over some signature. Generally, the signature represents a programming or a specification language and the terms are programs. The transition relation models the operational behavior of these terms.

Definition 2.2. Let Σ be a signature and A a set of labels. A (labeled) *transition relation* is a subset $\longrightarrow$ of $Tr(\Sigma, A)$ where $Tr(\Sigma, A) = T(\Sigma) \times A \times T(\Sigma)$. Elements (t, a, t') of a transition relation are written as $t \xrightarrow{a} t'$.

A transition relation is often defined by means of a *Transition System Specification* (*TSS*). PLOTKIN [14] defended the use of TSS's to give an operational semantics, and therefore a TSS is sometimes called an *operational semantics in Plotkin style*. The term TSS was first coined in [11] for a system in which rules had only positive premises. Negative premises were added in [10].

Definition 2.3. A *TSS* (*Transition System Specification*) is a triple $P = (\Sigma, A, R)$ with Σ a signature, A a set of labels and R a set of rules of the form:

$$\frac{\{t_k \xrightarrow{a_k} t'_k | k \in K\} \cup \{t_l \xrightarrow{b_l} | l \in L\}}{t \xrightarrow{a} t'}$$

with K, L (possibly infinite) index sets, $t_k, t'_k, t_l, t, t' \in \mathbb{T}(\Sigma)$, $a_k, b_l, a \in A$ $(k \in K,\ l \in L)$. An expression of the form $t \xrightarrow{a} t'$ is called a (*positive*) *literal*. $t \xrightarrow{a}$ is called a *negative literal*. φ, ψ, χ are used to range over literals. For any rule $r \in R$ the literals above the line are called the *premises* of r, notation $prem(r)$, and the literal below the line is called the *conclusion* of r, denoted as $conc(r)$. Furthermore, we write $pprem(r)$ for the set of positive premises of r and $nprem(r)$ for

the set of negative premises of r. A rule r is called *positive* if there are no negative premises, i.e. $nprem(r) = \emptyset$. A TSS is called *positive* if it has only positive rules. A rule is called an *axiom* if its set of premises is empty. An axiom $\frac{\emptyset}{t \xrightarrow{a} t'}$ is often written as $t \xrightarrow{a} t'$. The notions 'substitution', 'Var' and 'closed' extend to literals and rules as expected.

Throughout this article we use the following Transition System Specification scheme to illustrate the techniques we introduce. It describes the semantics of a small basic process language extended with priorities and abstraction. This combination has not been studied before due to the technical complications that are involved. Priorities are investigated in [1, 5, 6]. We follow the line set out by BAETEN, BERGSTRA and KLOP [1] who introduced a priority operator θ. For abstraction we take observation equivalence as introduced by MILNER [12] although technically we follow VAN GLABBEEK [9]. We base our example on $\mathrm{BPA}_{\delta\epsilon\tau}$, *Basic Process Algebra* with δ, ϵ and τ as introduced in [11], and extend it with recursion and priorities.

ϵ :	R1:	$\epsilon \xrightarrow{\surd} \delta$	
a :	R2:	$a \xrightarrow{a} \epsilon$ if $a \in Act_\tau$	
$+$:	R3.1:	$\frac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x'}$	R3.2: $\frac{y \xrightarrow{a} y'}{x + y \xrightarrow{a} y'}$
$\cdot$:	R4.1:	$\frac{x \xrightarrow{a} x'}{x \cdot y \xrightarrow{a} x' \cdot y}$ if $a \in Act_\tau$	R4.2: $\frac{x \xrightarrow{\surd} x' \quad y \xrightarrow{a} y'}{x \cdot y \xrightarrow{a} y'}$
θ :	R5.1:	$\frac{x \xrightarrow{a} x' \quad \forall b > a \; x \xrightarrow{b}\!\!\!\!\!\!/}{\theta(x) \xrightarrow{a} \theta(x')}$ if $a \in Act_\tau$	R5.2: $\frac{x \xrightarrow{\surd} x'}{\theta(x) \xrightarrow{\surd} \theta(x')}$
$\triangleleft$:	R6.1:	$\frac{x \xrightarrow{a} x' \quad \forall b > a \; y \xrightarrow{b}\!\!\!\!\!\!/}{x \triangleleft y \xrightarrow{a} x'}$ if $a \in Act_\tau$	R6.2: $\frac{x \xrightarrow{\surd} x'}{x \triangleleft y \xrightarrow{\surd} x'}$
τ_I :	R7.1:	$\frac{x \xrightarrow{a} x'}{\tau_I(x) \xrightarrow{a} \tau_I(x')}$ if $a \notin I$	R7.2: $\frac{x \xrightarrow{a} x'}{\tau_I(x) \xrightarrow{\tau} \tau_I(x')}$ if $a \in I$
recursion:	R8:	$\frac{t_X \xrightarrow{a} y}{X \xrightarrow{a} y}$ if $X \Leftarrow t_X \in E$	
τ-rules:	R9.1:	$a \xrightarrow{a} \tau$ if $a \in Act_\tau$	
	R9.2:	$\frac{x \xrightarrow{\tau} y \quad y \xrightarrow{a} z}{x \xrightarrow{a} z}$	R9.3: $\frac{x \xrightarrow{a} y \quad y \xrightarrow{\tau} z}{x \xrightarrow{a} z}$

Table 1: The operational semantics of $\mathrm{BPA}_{\delta\epsilon\tau}$ with priorities ($a \in Act_{\tau\surd}$, $b \in Act_\tau$).

Example 2.4 ($\mathrm{BPA}_{\delta\epsilon\tau}$ *with priorities*). We assume that we have a given set Act of actions that represent the basic activities that can be performed by processes. $Act_\tau = Act \cup \{\tau\}$ is a set of actions containing the symbol τ representing internal or hidden activity. Moreover, we assume a partial ordering $<$ on Act_τ, which we call the *priority relation*: actions higher in the ordering have a higher priority than actions lower in the ordering. We assume that $<$ is backwardly well-founded,

i.e. the inverse of $<$ constitutes a well-founded ordering.

Our signature contains a constant a for each action $a \in Act_\tau$. Moreover, we have the two constants δ and ϵ. δ is called *inaction* (or *deadlock*) and it represents the process that cannot do anything at all. ϵ is called the *empty process* which cannot do anything but terminate. *Sequential composition* is written as '$\cdot$' and *alternative composition* is denoted by $+$. We often leave out the '$\cdot$' and assume that '$\cdot$' binds stronger than $+$. Actions can be renamed to τ by *abstraction operators* τ_I ($I \subseteq Act$).

For recursion it is assumed that there is some given set Ξ of *process names*. Each process name $X \in \Xi$ is treated as a constant in the signature. Furthermore, we assume that a set E of *process declarations* is given. For each process name X in Ξ there is a declaration $X \Leftarrow t_X \in E$ where t_X is a closed term over the signature. Terms that do not contain process names are called *recursion free*.

The remaining operators in the signature deal with priorities. The *priority operator* θ acts as a sieve: $\theta(x)$ only allows those actions from x that have highest priority. For the axiomatization of $\mathrm{BPA}_{\delta\epsilon\tau}$ with priorities, which is given in section 8, we need the *unless* operator $\lhd$, which was introduced in [1]. This operator is applied on two operands and only allows an action in its left-hand side provided the right-hand side cannot do any action with higher priority.

When $(Act_\tau, <)$ and (Ξ, E) are fixed, we obtain a TSS which is an *instance* of $\mathrm{BPA}_{\delta\epsilon\tau}$ with priorities. Such an instance will be denoted as $P_\theta = (\Sigma_\theta, A_\theta, R_\theta)$. The signature $\Sigma_\theta = (F_\theta, rank_\theta)$ is described above. The labels in A_θ are exactly those in Act_τ together with one special symbol $\surd$: if a process term t can perform a $\surd$-step, i.e. $t \xrightarrow{\surd} t'$, this means that t has an option to terminate.

The rules in R_θ are given in table 1. Here, the action a ranges over $Act_{\tau\surd} = Act_\tau \cup \{\surd\}$ and b ranges over Act_τ. In rules R5.1 and R6.1 we use the notation $\forall b > a\ x \xrightarrow{b}\!\!\!\!\!/\ $ which means that for all b with higher priority than a, there is a negative premise $x \xrightarrow{b}\!\!\!\!\!/\ $. We think that the remaining rules are self explanatory.

3 Transition relations for TSS's

We have introduced TSS's as a formalism for specifying transition relations. Thus a most fundamental question is which transition relation is actually defined by a TSS. In this section we outline some answers proposed in the literature for several classes of TSS's. Then we show an instance of our running example for which these techniques are not adequate.

First, a link between the transitions in a transition relation and the literals in TSS's is established.

Definition 3.1. Let $\longrightarrow$ be a transition relation. A positive ground literal ψ *holds* in $\longrightarrow$ or ψ is *valid* in $\longrightarrow$, notation $\longrightarrow \models \psi$, if the transition $\psi \in \longrightarrow$. A negative ground literal $t \xrightarrow{a}\!\!\!\!\!/\ $ *holds* in $\longrightarrow$, notation $\longrightarrow \models t \xrightarrow{a}\!\!\!\!\!/\ $, if for no $t' \in T(\Sigma)$ the transition $t \xrightarrow{a} t' \in \longrightarrow$. For a set of ground literals Ψ, we write $\longrightarrow \models \Psi$ iff $\forall \psi \in \Psi$: $\longrightarrow \models \psi$.

At least one may require that a transition relation associated with a TSS P obeys the rules of P, i.e. if the premises of a ground instance of a rule in P are valid in $\longrightarrow$, then the conclusion is also valid in $\longrightarrow$. (In terms of logic: the rules of P, interpreted as implications, are true in $\longrightarrow$).

Definition 3.2. Let $P = (\Sigma, A, R)$ be a TSS and let $\longrightarrow \subseteq Tr(\Sigma, A)$ be a transition relation. $\longrightarrow$ is a *model* of P if:

$$\psi \in \longrightarrow \quad \Leftarrow \quad \exists r \in R \text{ and } \exists \sigma : V \to T(\Sigma) \text{ such that : } \begin{cases} \longrightarrow \models prem(\sigma(r)) \text{ and} \\ conc(\sigma(r)) = \psi. \end{cases}$$

On the other hand, a transition ψ should not be incorporated in the transition relation $\longrightarrow$ of a TSS P unless there is a good reason to do so, namely a rule in P with valid premises in $\longrightarrow$ concluding ψ.

Definition 3.3. Let $P = (\Sigma, A, R)$ be a TSS. Let $\longrightarrow \subseteq Tr(\Sigma, A)$ be a transition relation. $\longrightarrow$ is *supported* by P if:

$$\psi \in \longrightarrow \quad \Rightarrow \quad \exists r \in R \text{ and } \exists \sigma : V \to T(\Sigma) \text{ such that : } \begin{cases} \longrightarrow \models prem(\sigma(r)) \text{ and} \\ conc(\sigma(r)) = \psi. \end{cases}$$

Combining the previous definitions, we get:

Definition 3.4. Let $P = (\Sigma, A, R)$ be a TSS. Let $\longrightarrow \subseteq Tr(\Sigma, A)$ be a transition relation. $\longrightarrow$ is a *supported model* of P if $\longrightarrow$ is supported by P and $\longrightarrow$ is a model of P.

The notion of $\longrightarrow$ being a supported model of P was introduced in [2] as '$\longrightarrow$ *agrees with* P'. Although the transition relation associated with a TSS should certainly be a supported model of it, the notion of supportedness is generally not sufficient to exclude all superfluous transitions from the transition relation. For positive TSS's this shortcoming is easily remedied by associating with a TSS P the *least* transition relation (w.r.t. set inclusion) that is a model of P.

Definition 3.5. The transition relation $\longrightarrow_P$ *associated with* a positive TSS P is the least model of P w.r.t. set inclusion.

Traditionally ([11, 14]) an alternative but equivalent definition of the transition relation associated with a positive TSS was given, based on the *provability* of transitions. For TSS's with negative premises it is much more difficult to find an appropriate associated transition relation as is shown by the following example.

Example 3.6. Suppose we have a TSS P with one constant f, two labels a and b and the following rules:

$$\frac{f \xrightarrow{a} f}{f \xrightarrow{a} f} \qquad \frac{f \xrightarrow{a} /}{f \xrightarrow{b} f}.$$

We would like P to define the transition relation $\longrightarrow_P = \{f \xrightarrow{b} f\}$. However, P has exactly two minimal models, $\{f \xrightarrow{a} f\}$ and $\{f \xrightarrow{b} f\}$, which are both supported.

Thus in the presence of negative premises there may be several minimal models, some of them may be supported. So other characterizations for associated transition relations must be sought. The notion of provability also needs a revision, as it is not a priori clear how the negative premises of a rule must be proved.

Similar problems concerning negative premises have been studied in the context of logic programming. A first solution proposed there introduced the notion of (local) stratification [16]. Here we follow [10], where this notion was tailored for TSS's.

Definition 3.7. Let $P = (\Sigma, A, R)$ be a TSS. A function $S : Tr(\Sigma, A) \to \alpha$, where α is an ordinal, is called a *stratification* of P if for every rule $r \in R$ and every substitution $\sigma : V \to T(\Sigma)$ it holds that:

$$\begin{array}{l} \text{for all } \psi \in pprem(\sigma(r)) : \ S(\psi) \leq S(conc(\sigma(r)) \text{ and} \\ \text{for all } t \xrightarrow{a} / \in nprem(\sigma(r)) \text{ and } t' \in T(\Sigma) : \ S(t \xrightarrow{a} t') < S(conc(\sigma(r))). \end{array}$$

If P has a stratification, we say that P is *stratified.*

The stratification guarantees that the validity of no literal depends on the validity of its negation.

Example 3.8. The TSS of example 3.6 can be stratified by a stratification S as follows:

$$S(f \xrightarrow{a} f) = 0 \text{ and } S(f \xrightarrow{b} f) = 1.$$

Each positive transition system specification is trivially stratified by putting all positive literals in stratum 0. In [10] it is shown that $\mathrm{BPA}_{\epsilon\delta}$ with priorities and renaming but without abstraction is stratified under some appropriate conditions.

How a transition relation $\longrightarrow_{P,S}$ is associated to a stratified TSS P is defined in [10]. Although the stratification technique is often applicable, there are examples of TSS's that have an intuitive meaning while not being stratified. One such example is $\mathrm{BPA}_{\delta\epsilon\tau}$ with priorities.

Example 3.9. Suppose we have an instance P_θ of $\mathrm{BPA}_{\delta\epsilon\tau}$ with priorities based on a set of actions Act containing at least two elements a and b such that $a < b$. Consider for arbitrary terms t and u the following instances of rules which prove that for any stratification S of P_θ it should hold that:

$$S(t\xrightarrow{b}\tau_{\{a\}}(\theta(u))) < \quad \text{by} \quad \frac{t\xrightarrow{a}u \;\; t\not\xrightarrow{b}}{\theta(t)\xrightarrow{a}\theta(u)} \qquad \text{(R5.1)}$$

$$S(\theta(t)\xrightarrow{a}\theta(u)) \leq \quad \text{by} \quad \frac{\theta(t)\xrightarrow{a}\theta(u)}{\tau_{\{a\}}(\theta(t))\xrightarrow{\tau}\tau_{\{a\}}(\theta(u))} \qquad \text{(R7.2)}$$

$$S(\tau_{\{a\}}(\theta(t))\xrightarrow{\tau}\tau_{\{a\}}(\theta(u))) \leq \quad \text{by} \quad \frac{t\xrightarrow{b}\tau_{\{a\}}(\theta(t)) \;\; \tau_{\{a\}}(\theta(t))\xrightarrow{\tau}\tau_{\{a\}}(\theta(u))}{t\xrightarrow{b}\tau_{\{a\}}(\theta(u))} \qquad \text{(R9.3)}$$

$$S(t\xrightarrow{b}\tau_{\{a\}}(\theta(u))).$$

Of course, such a stratification cannot exist.

4 TSS's and their associated transition relations

So far no meaning has been given to TSS's that are not stratified. There are however TSS's, like $\mathrm{BPA}_{\delta\epsilon\tau}$ with priorities, that seem to be perfectly meaningful while not being stratified. This brings us back to the fundamental question what transition relation should be associated with a TSS. Our answer is essentially that the transition relation must be the unique *stable model* in the sense of logic programming [8]. We strongly believe that any TSS that has no unique stable transition relation does not have a proper meaning.

The definition of a stable transition relation is intuitively as follows. Our first observation is that positive and negative premises in a rule of a TSS P have a different status. In order to prove the conclusion of a rule, the positive premises of the rule must be proved from P. However, as P contains only rules defining which literals hold, but not which literals do not hold, negative premises must be treated differently.

Conceptually, $t\not\xrightarrow{a}$ holds by default, i.e. if for no t': $t\xrightarrow{a}t'$ can be proved. But we are still trying to determine which literals can be proved. So instead of an immediate characterization of the set of provable literals $\longrightarrow$, we have an equation with this set both on the left and on the right side, namely:

> $\longrightarrow$ equals the set of literals that are provable by those rules of the TSS of which the negative premises hold in $\longrightarrow$.

This equation does not give us a means to compute the transition relation $\longrightarrow$, but we can easily check whether a given transition relation satisfies our criterion.

We now formalize these ideas. In section 4, 5 and 6 we use only ground TSS's, i.e. we identify a set of rules R with the set of ground instances of R.

Definition 4.1. Let $P = (\Sigma, A, R)$ be a TSS. Let $\longrightarrow \subseteq Tr(\Sigma, A)$.

$$Strip(P, \longrightarrow) = (\Sigma, A, Strip(R, \longrightarrow))$$

where

$$Strip(R, \longrightarrow) = \{r' | \exists r \in R : \ \longrightarrow \models nprem(r) \text{ and } r' = \frac{pprem(r)}{conc(r)}\}.$$

Given a transition relation $\longrightarrow$, the function *Strip* removes all rules in R that have negative premises that do not hold in $\longrightarrow$. Furthermore, it drops the negative premises from the remaining rules. The following lemma is therefore obvious.

Lemma 4.2. *Let $P = (\Sigma, A, R)$ be a TSS and let $\longrightarrow \subseteq Tr(\Sigma, A)$ be a transition relation. Then $Strip(P, \longrightarrow)$ is a positive TSS.*

Using the fact that the notion of provability is already captured in the definition of the transition relation associated with a positive TSS, we can now easily formalize the previously stated equation.

Definition 4.3 *(Stable transition relation).* Let $P = (\Sigma, A, R)$ be a TSS. A transition relation $\longrightarrow \subseteq Tr(\Sigma, A)$ is *stable* for P if $\longrightarrow = \longrightarrow_{Strip(P, \longrightarrow)}$.

Remark 4.4. In general, for a TSS P there may be 0, 1 or more transition relations that are stable for P, e.g.

$$0: \ \frac{f \overset{a}{\not\longrightarrow}}{f \overset{a}{\longrightarrow} f}$$

$$1: \ \frac{f \overset{a}{\not\longrightarrow} \quad f \overset{b}{\longrightarrow} f}{f \overset{a}{\longrightarrow} f} \qquad [\longrightarrow = \emptyset]$$

$$2: \ \frac{f \overset{a}{\not\longrightarrow}}{f \overset{b}{\longrightarrow} f} \quad \frac{f \overset{b}{\not\longrightarrow}}{f \overset{a}{\longrightarrow} f} \quad [\longrightarrow = \{f \overset{a}{\longrightarrow} f\} \text{ or } \longrightarrow = \{f \overset{b}{\longrightarrow} f\}]$$

We do not have any idea as to which transition relations should be associated with the first TSS, nor do we know which one of the two transition relations of the third TSS should be preferred. In fact we think that there are no satisfying answers to those questions. Thus we propound the following definition.

Definition 4.5. Let P be a TSS. If there is a unique transition relation $\longrightarrow$ stable for P, then $\longrightarrow$ is the transition relation *associated with* P.

In order to avoid confusion, we do not again introduce the notation $\longrightarrow_P$: until section 7 this notation remains reserved for stratified TSS's.

Remark 4.6. If P is positive, then for every transition relation $\longrightarrow$, $Strip(P, \longrightarrow) = P$, thus $\longrightarrow_P$ is the unique transition relation that is stable for P. Hence, this definition of 'associated with' coincides with the previously given definition for positive TSS's. In section 6 we show that our choice also extends the definition of 'associated with' for stratified TSS's.

Our choice that a transition relation must be stable for a TSS is also a refinement of the requirement that a transition relation must be a supported and minimal model of it (Cf. [8], theorem 1). We now apply the notion *is stable for* to our running example.

Example 4.7. Consider P_θ with at least two actions a and b such that $a > b$ and a process name X with the recursive definition

$$X \Leftarrow \theta(\tau_{\{b\}}(X) \cdot a + b).$$

It can be shown that there is no transition relation $\longrightarrow$ that is stable for P_θ. If one assumes that $\longrightarrow \models \tau_{\{b\}}(X) \xrightarrow{\surd} t$ for some t or that $\longrightarrow \models \tau_{\{b\}} \overset{a}{\not\longrightarrow}$, then a contradiction follows. If one assumes that $\longrightarrow \models \tau_{\{b\}}(X) \overset{\surd}{\not\longrightarrow}$ and that $\longrightarrow \models \tau_{\{b\}}(X) \xrightarrow{a} t$ for some t, then it appears that for no t, $\tau_{\{b\}}(X) \xrightarrow{a} t$ is provable from $Strip(P_\theta, \longrightarrow)$.

5 Reducing TSS's

We now present a technique that can be useful for proving that a certain TSS has a unique stable transition relation. This technique is inspired by the *well-founded models* that are introduced in [7]. First we construct a 3-valued 'interpretation' for a TSS P, partitioning the set of transitions in three groups: those that are certainly true, those of which the truth is unknown and those that are certainly not true. We apply this information to *reduce* P to another TSS with exactly the same stable transition relations as P. In this new TSS, the truth or falsity of more literals may become certain. Repeated reduction may lead to complete information: the unique stable transition relation.

If in the next definition $\longrightarrow_{true}$ contains transitions that certainly hold and $\longrightarrow_{pos}$ contains all transitions that possibly hold, then rules with certainly wrong premises are removed and in the remaining rules all premises that certainly hold are dropped.

Definition 5.1. Let $P = (\Sigma, A, R)$ be a TSS. Let $\longrightarrow_{true}, \longrightarrow_{pos} \subseteq Tr(\Sigma, A)$ be transition relations.

$$Reduce(P, \longrightarrow_{true}, \longrightarrow_{pos}) = (\Sigma, A, Reduce(R, \longrightarrow_{true}, \longrightarrow_{pos})),$$

where

$$Reduce(R, \longrightarrow_{true}, \longrightarrow_{pos}) = \{r' | \exists r \in R : \longrightarrow_{true} \models nprem(r),\ \longrightarrow_{pos} \models pprem(r) \text{ and}$$

$$r' = \frac{\{\psi \in pprem(r) | \longrightarrow_{true} \not\models \psi\} \cup \{\psi \in nprem(r) | \longrightarrow_{pos} \not\models \psi\}}{conc(r)}\}.$$

Thus the reduction of a rule consists of two phases. First it is checked that the premises are *possibly* true. For positive premises this is straightforward: $t \xrightarrow{a} t'$ is possibly true if $t \xrightarrow{a} t' \in \longrightarrow_{pos}$. Hence the condition $\longrightarrow_{pos} \models pprem(r)$. A negative premise $t \overset{a}{\not\longrightarrow}$ is possibly true if it is not certain that t can perform an a-step, i.e. for no t' it is certain that $t \xrightarrow{a} t'$ holds. Thus $t \overset{a}{\not\longrightarrow}$ is possibly true if for no t', $t \xrightarrow{a} t' \in \longrightarrow_{true}$. Hence the condition $\longrightarrow_{true} \models nprem(r)$.

If indeed the premises of the rule are possibly true, then the premises that are *certainly* true are removed. A positive premise $t \xrightarrow{a} t'$ is certainly true if $t \xrightarrow{a} t' \in \longrightarrow_{true}$. A negative premise $t \overset{a}{\not\longrightarrow}$ is certainly true if t cannot possibly perform an a-step, i.e. for no t': $t \xrightarrow{a} t'$ is possible. Thus $t \overset{a}{\not\longrightarrow}$ is certainly true if $\longrightarrow_{pos} \models t \overset{a}{\not\longrightarrow}$. We shall always use definition 5.1 with $\longrightarrow_{true} \subseteq \longrightarrow_{pos}$.

Remark 5.2. Note that $Reduce(R, \longrightarrow, \longrightarrow)$ differs from $Strip(R, \longrightarrow)$. In $Strip(R, \longrightarrow)$ only negative premises are checked, yielding a positive TSS; in $Reduce(R, \longrightarrow, \longrightarrow)$ all premises are checked, resulting in a TSS consisting solely of rules without premises.

The 3-valued interpretation required is obtained by means of two positive TSS's: $True(P)$ and $Pos(P)$. $True(P)$ determines the transitions that are certainly true: the transitions that can be proved with positive rules only. $Pos(P)$ determines the transitions that are possibly true, i.e.

true or unknown. These are the transitions that can be proved ignoring negative premises. Thus $Pos(P)$ is obtained from P by removing all negative premises of the rules.

Definition 5.3. Let $P = (\Sigma, A, R)$ be a TSS.

- $True(P) = (\Sigma, A, True(R))$ where $True(R) = \{r \in R | nprem(r) = \emptyset\}$.
- $Pos(P) = (\Sigma, A, Pos(R))$ where $Pos(R) = \{r' | \exists r \in R : r' = \frac{pprem(r)}{conc(r)}\}$.

Because after the reduction of P the truth or falsity of more literals may become certain, it is worthwhile to iterate the reduction process; if necessary even transfinitely many reduction steps may be considered.

Definition 5.4. Let $P = (\Sigma, A, R)$ be a TSS. For every ordinal α, the α-reduction of P, notation $Red^\alpha(P)$, is recursively defined as follows:

- $Red^0(P) = (\Sigma, A, R_{ground})$ where R_{ground} is the set of all ground instances of rules in R,
- $Red^\alpha(P) = Reduce(P, \bigcup_{\beta<\alpha} \longrightarrow_{True(Red^\beta(P))}, \bigcap_{\beta<\alpha} \longrightarrow_{Pos(Red^\beta(P))})$.

Thus in contrast with [7] and general practice in logic programming, our operator maps TSS's to TSS's rather than interpretations to interpretations. This allows us in section 6 to combine reduction with stratification: as soon as the reduced TSS is stratified, no further reduction is needed.

Our hope is that after sufficiently many reductions we obtain a positive TSS. If this is the case, then our method has succeeded: the transition relation of this positive TSS is the unique transition relation that is stable for the original one. (Example 7.5 shows that the converse is not true: a TSS having a unique stable transition relation need not reduce to a positive TSS.)

Theorem 5.5 *(Soundness of reduction).* *Let $P = (\Sigma, A, R)$ be a TSS and let $\longrightarrow \subseteq Tr(\Sigma, A)$. For all ordinals α we have:*

$$\longrightarrow \text{ is stable for } P \Leftrightarrow \longrightarrow \text{ is stable for } Red^\alpha(P).$$

Corollary 5.6 *(Cf. [7], corollary 6.2).* *If P reduces to a positive TSS, i.e. $Red^\alpha(P)$ is positive for some α, then $\longrightarrow_{Red^\alpha(P)}$ is associated with P.*

6 Reduction and stratification

We now have two independent methods for associating a transition relation with a TSS with negative premises: reduction and stratification. Three questions arise:

- if both methods are applicable, is their result the same?
- is one method (strictly) stronger than the other?
- is it useful to combine the two methods?

In this section we answer these questions affirmatively. For a stratified TSS P, the relation $\longrightarrow_P$ as defined in section 3 is stable for P. Furthermore, repeatedly reducing a stratified TSS yields a positive TSS. Thus $\longrightarrow_P$ is the unique transition relation that is stable for P. This is also the answer to our second question: reduction is indeed stronger than stratification (that it is strictly stronger is easily seen by the second TSS in remark 4.4).

Theorem 6.1 *(Cf. [8], corollary 1 and [7], theorem 6.3). Let $P = (\Sigma, A, R)$ be a TSS with stratification $S : Tr(\Sigma, A) \to \alpha$. Then $Red^{\alpha}(P)$ is a positive TSS and $\longrightarrow_P = \longrightarrow_{Red^{\alpha}(P)}$ is associated with P.*

So it seems that there is no point in combining the two methods: the result could not be stronger than reduction alone. However, for practical purposes the combination appears to be valuable, due to the fact that the existence of a stratification is generally easier to demonstrate.

Corollary 6.2 *(Combining reduction and stratification). Let $P = (\Sigma, A, R)$ be a TSS and suppose that for some ordinals α and β, $S : Tr(\Sigma, A) \to \alpha$ is a stratification of $Red^{\beta}(P)$. Then $Red^{\alpha+\beta}(P)$ is a positive TSS and $\longrightarrow_{Red^{\alpha+\beta}(P)} = \longrightarrow_{Red^{\beta}(P)}$ is associated with P.*

In the remainder of this section we apply this corollary to show that a transition relation is associated with an instance P_θ of $\mathrm{BPA}_{\delta\epsilon\tau}$ with priorities, provided that two conditions hold:

1. The abstraction operator τ_I does not occur in process terms. The reason for this condition was already shown in example 4.7. This is conform the standard practise in process algebra.
2. There is no $a \in Act$ such that $\tau < a$ (cf. [18] where it is argued that $\tau > a$ for all actions a seems the most 'intuitive' choice).

Theorem 6.3. *If for all $(X \Leftarrow t_X) \in E$: $\tau_I(\cdot)$ does not occur in t_X and for all $a \in Act$ it does not hold that $\tau < a$, then there is a transition relation associated with P_θ.*

Proof. We show that P_θ is stratified after one reduction step. To this end we formulate a useful property of $Red^1(P_\theta)$. Define $N : T(\Sigma_\theta) \to \mathbb{N}$ by:

$$\begin{aligned}
&N(a) = N(\epsilon) = N(\delta) = N(\tau) = N(X) = 0 \quad (X \in \Xi \text{ and } a \in Act),\\
&N(x + y) = N(x \cdot y) = N(x \lhd y) = max(N(x), N(y)),\\
&N(\theta(x)) = N(x),\\
&N(\tau_I(x)) = N(x) + 1.
\end{aligned}$$

We show that it is not possible to prove in P_θ a literal $t \xrightarrow{a} u$ when $N(t) < N(u)$ (i.e. the 'N-complexity' of a process, the depth of nestings of $\tau_I(\cdot)$'s in it, cannot increase by performing an action).

Fact 1. *For all $a \in A_\theta$ we have:* $t \xrightarrow{a} u \in \longrightarrow_{Pos(P_\theta)} \;\Rightarrow\; N(t) \geq N(u)$.

For example, the literal $t \xrightarrow{b} \tau_{\{a\}}(\theta(t))$ used in example 3.9 to make $t \xrightarrow{b} \tau_{\{a\}}(\theta(u))$ depend negatively on itself, is not possible. Based on this definition of N we define the preorder $\leq$ on pairs of literals by:

$$(t \xrightarrow{a} u) \leq (t' \xrightarrow{b} u') \text{ iff } \begin{cases} N(t) < N(t') \text{ or} \\ N(t) = N(t') \text{ and } (a = \tau, \surd),\ a > b, \text{ or } a = b. \end{cases}$$

For some ordinal α we can now define a function $S : Tr(\Sigma_\theta, A_\theta) \to \alpha$ obtained by transforming the preorder $\leq$ into a complete well-founded ordering:

$$\begin{aligned}
&\varphi \approx \psi \text{ iff } \varphi \leq \psi \text{ and } \psi \leq \varphi,\\
&\varphi \approx \psi \;\Rightarrow\; S(\varphi) = S(\psi),\\
&\varphi \leq \psi \text{ and not } \varphi \approx \psi \;\Rightarrow\; S(\varphi) < S(\psi).
\end{aligned}$$

(We do not need a more precise definition of S; since such a definition necessarily depends on the size of the set Act, we omit it).

Fact 2. *S is a stratification of $Reduce(P_\theta, \longrightarrow_{True(P_\theta)}, \longrightarrow_{Pos(P_\theta)})$.*

Using corollary 6.2 $\longrightarrow_{Reduce(P_\theta, \longrightarrow_{True(P_\theta)}, \longrightarrow_{Pos(P_\theta)})} = \longrightarrow_{Red^1(P_\theta)}$ is associated with P_θ. □

7 The *ntyft/ntyxt*-format and bisimulation congruence

We have defined the meaning of a TSS as its associated transition relation and shown how to arrive at this transition relation. Now we switch to the study of properties of transition relations as consequences of properties of their defining TSS's.

An important question (e.g. in process verification) is whether two terms denote the 'same' process. Many process equivalences based on transition relations have been proposed, of which strong bisimulation equivalence is most often used [12, 13]. In this section some relations between TSS's and strong bisimulation equivalence are studied.

Definition 7.1. Let P be a TSS with associated transition relation $\longrightarrow_P$. A relation R is a *strong bisimulation relation* based on P if it satisfies:

- whenever tRu and $t \xrightarrow{a}_P t'$ then, for some $u' \in T(\Sigma)$, we have $u \xrightarrow{a}_P u'$ and $t'Ru'$,
- whenever tRu and $u \xrightarrow{a}_P u'$ then, for some $t' \in T(\Sigma)$, we have $t \xrightarrow{a}_P t'$ and $t'Ru'$.

Two terms $t, u \in T(\Sigma)$ are *(P-)bisimilar*, notation $t \leftrightarrows_P u$, if there is a strong bisimulation relation R based on P such that tRu. Note that $\leftrightarrows_P$, the strong bisimulation equivalence *induced* by P, is an equivalence relation.

A desirable property for TSS's is that the induced strong bisimulation equivalence is a congruence. In [11] this led to the observation that if a (positive) TSS is in the so-called *tyft/tyxt*-format then this is the case. In [10] this result was extended to stratified TSS's. In order to express the fact that negative premises are allowed, *n*'s were added to the name of the format, obtaining the *ntyft/ntyxt*-format. In this section we show that even for TSS's that are positive after reduction, bisimulation is a congruence if the TSS is in *ntyft/ntyxt*-format.

Definition 7.2. Let $\Sigma = (F, rank)$ be a signature. Let $P = (\Sigma, A, R)$ be a TSS. A rule $r \in R$ is in *ntyft-format* if it has the form:

$$\frac{\{t_k \xrightarrow{a_k} y_k | k \in K\} \cup \{t_l \xrightarrow{b_l} | l \in L\}}{f(x_1, ..., x_{rank(f)}) \xrightarrow{a} t}$$

with K and L (possibly infinite) index sets, y_k, x_i $(1 \leq i \leq rank(f))$ all different variables, $a_k, b_l, a \in A$, $f \in F$ and $t_k, t_l, t \in \mathbb{T}(\Sigma)$. A rule $r \in R$ is in *ntyxt-format* if it fits:

$$\frac{\{t_k \xrightarrow{a_k} y_k | k \in K\} \cup \{t_l \xrightarrow{b_l} | l \in L\}}{x \xrightarrow{a} t}$$

with K, L (possibly infinite) index sets, y_k, x all different variables, $a_k, b_l, a \in A$, t_k, t_l and $t \in \mathbb{T}(\Sigma)$. P is in *ntyft/ntyxt*-format if all its rules are either in *ntyft*- or in *ntyxt*-format.

As in [10, 11], we need the following well-foundedness condition in order to prove the congruence theorem.

Definition 7.3 *(Well-foundedness).* Let $P = (\Sigma, A, R)$ be a TSS. Let $S = \{t_k \xrightarrow{a_k} t'_k | k \in K\} \subseteq \mathbb{T}(\Sigma) \times A \times \mathbb{T}(\Sigma)$ be a set of positive literals over Σ and A. The *variable dependency graph* of S is a directed (unlabeled) graph VDG with:

- Nodes: $\bigcup_{k \in K} Var(t_k \xrightarrow{a_k} t'_k)$,
- Edges: $\{< x, y > | x \in Var(t_k),\ y \in Var(t'_k)$ for some $k \in K\}$.

S is called *well-founded* if any backward chain of edges in the variable dependency graph is finite. A rule is called *well-founded* if its set of positive premises is well-founded. A TSS is called *well-founded* if all its rules are well-founded.

Theorem 7.4 *(Congruence theorem). Let P be a well-founded TSS in ntyft/ntyxt-format that is positive after reduction. Then $\leftrightarrows_P$ is a congruence.*

The next example shows that the requirement in the congruence theorem that the TSS P must be positive after reduction is really needed. We give a TSS in *ntyft/ntyxt*-format that has a unique stable transition relation but that is *not* positive after reduction and for which bisimulation is *not* a congruence.

Example 7.5. Let $P = (\Sigma, A, R)$ be a TSS where Σ contains constants c_1 and c_2 and a unary function f. The actions in A are a, b_1, b_2 and the rules are the following:

$$\text{E1:}\ \ c_1 \xrightarrow{a} c_1 \qquad \text{E2:}\ \ c_2 \xrightarrow{a} c_2$$

$$\text{E3:}\ \ \frac{x \xrightarrow{a} y \quad f(x) \overset{b_1}{\not\longrightarrow} \quad f(c_1) \overset{b_2}{\not\longrightarrow}}{f(x) \xrightarrow{b_2} c_2} \qquad \text{E4:}\ \ \frac{x \xrightarrow{a} y \quad f(x) \overset{b_2}{\not\longrightarrow} \quad f(c_2) \overset{b_1}{\not\longrightarrow}}{f(x) \xrightarrow{b_1} c_1}.$$

$Red^1(P)$ is a TSS with the following rules:

$$\text{E1}'\text{:}\ \ c_1 \xrightarrow{a} c_1 \qquad \text{E2}'\text{:}\ \ c_2 \xrightarrow{a} c_2$$

$$\text{E3}'\text{:}\ \ \frac{f(c_1) \overset{b_1}{\not\longrightarrow} \quad f(c_1) \overset{b_2}{\not\longrightarrow}}{f(c_1) \xrightarrow{b_2} c_2} \qquad \text{E3}''\text{:}\ \ \frac{f(c_2) \overset{b_1}{\not\longrightarrow} \quad f(c_1) \overset{b_2}{\not\longrightarrow}}{f(c_2) \xrightarrow{b_2} c_2}$$

$$\text{E4}'\text{:}\ \ \frac{f(c_1) \overset{b_2}{\not\longrightarrow} \quad f(c_2) \overset{b_1}{\not\longrightarrow}}{f(c_1) \xrightarrow{b_1} c_1} \qquad \text{E4}''\text{:}\ \ \frac{f(c_2) \overset{b_2}{\not\longrightarrow} \quad f(c_2) \overset{b_1}{\not\longrightarrow}}{f(c_2) \xrightarrow{b_1} c_1}.$$

Further reduction of P is not possible. However, we observe that both in E3′ and E4″ the conclusion denies the second premise. Therefore, a transition relation that is stable for P must deny the first premise of E3′ and of E4″, i.e. it must contain $f(c_1) \xrightarrow{b_1} t_1$ and $f(c_2) \xrightarrow{b_2} t_2$ for some t_1 and t_2. The only candidates that might be provable are $f(c_1) \xrightarrow{b_1} c_1$ and $f(c_2) \xrightarrow{b_2} c_2$. Indeed they are provable from E3″ and E4′ (as blocking E3′ and E4″ implies $f(c_1) \overset{b_2}{\not\longrightarrow}$ and $f(c_2) \overset{b_1}{\not\longrightarrow}$), so $\{c_1 \xrightarrow{a} c_1,\ c_2 \xrightarrow{a} c_2,\ f(c_1) \xrightarrow{b_1} c_1,\ f(c_2) \xrightarrow{b_2} c_2\}$ is the unique transition relation that is stable for P. Now it is obvious that $c_1 \leftrightarrows_P c_2$, but not $f(c_1) \leftrightarrows_P f(c_2)$, so $\leftrightarrows_P$ is not a congruence.

8 An axiomatization of priorities with abstraction

This last section is devoted to our running example. We consider an instance $P_\theta = (\Sigma_\theta, A_\theta, R_\theta)$ of $\mathrm{BPA}_{\delta\epsilon\tau}$ with priorities such that for all $(X \Leftarrow t_X) \in E$: $\tau_I(\cdot)$ does not occur in t_X and for all $a \in Act$ it does not hold that $\tau < a$. By theorem 6.3 P_θ has an associated transition relation $\xrightarrow{}_{P_\theta}$.

In table 2 we list the axiom set $\mathrm{BPA}^\theta_{\delta\epsilon\tau}$ for strong bisimulation equivalence induced by P_θ. This axiom system consists of a straightforward assembly of existing axioms [1, 12], adding only the axiom P3 showing the interaction between $\lhd$ and τ. Nevertheless, as far as we know, this straightforward compilation has not been justified in bisimulation semantics. Only in [18] τ and θ have been combined using an isomorphic embedding.

Definition 8.1. Let $\Sigma = (F, rank)$ be a signature and let Eq be a set of axioms over Σ. Let $R_{Eq} \subseteq T(\Sigma) \times T(\Sigma)$ be the smallest congruence relation satisfying that $tR_{Eq}u$ if $t = u$ is a ground instance of an axiom in Eq. For terms $t, u \in T(\Sigma)$, we say that Eq *proves* $t = u$, notation $Eq \vdash t = u$, if $tR_{Eq}u$.

$\mathrm{BPA}^\theta_{\delta\epsilon\tau}$ is sound and for recursion free process terms also complete with respect to $\leftrightarrows_{P_\theta}$.

$x+(y+z)=(x+y)+z$	A1	$a\tau=a$	T1
$x+y=y+x$	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	$\theta(\epsilon)=\epsilon$	THE
$\delta x=\delta$	A7	$\theta(\delta)=\delta$	THD
$\epsilon x=x$	A8	$\theta(ax)=a\theta(x)$	TH1
$x\epsilon=x$	A9	$\theta(x+y)=\theta(x)\lhd y+\theta(y)\lhd x$	TH2
$\epsilon\lhd x=\epsilon$	PE1	$\tau_I(\epsilon)=\epsilon$	TIE
$x\lhd\epsilon=x$	PE2	$\tau_I(\delta)=\delta$	TID
$\delta\lhd x=\delta$	PD1	$\tau_I(a)=a$ if $a\notin I$	TI1
$x\lhd\delta=x$	PD2	$\tau_I(a)=\tau$ if $a\in I$	TI2
$ax\lhd by=\delta$ if $(a<b)$	P1	$\tau_I(x+y)=\tau_I(x)+\tau_I(y)$	TI3
$ax\lhd cy=ax$ if $\neg(a<c)$	P2	$\tau_I(xy)=\tau_I(x)\tau_I(y)$	TI4
$ax\lhd\tau y=ax\lhd y$ if $\neg(a<\tau)$	P3		
$x\lhd(y+z)=(x\lhd y)\lhd z$	P4		
$(x+y)\lhd z=x\lhd z+y\lhd z$	P5		

Table 2: The axiom set $\mathrm{BPA}^{\theta}_{\delta\epsilon\tau}$ ($a,b\in Act_\tau$ and $c\in Act$).

Theorem 8.2. *(Soundness) Let $t,u\in T(\Sigma_\theta)$. It holds that:*

$$\mathrm{BPA}^{\theta}_{\delta\epsilon\tau}\vdash t=u \;\Rightarrow\; t\leftrightarrows_{P_\theta}u.$$

Theorem 8.3. *(Completeness) Let $t,u\in T(\Sigma_\theta)$ be recursion free. It holds that:*

$$t\leftrightarrows_{P_\theta}u \;\Rightarrow\; \mathrm{BPA}^{\theta}_{\delta\epsilon\tau}\vdash t=u$$

The soundness and completeness proofs can be carried out by induction on proof trees (standard for positive TSS's) of the 'stripped' version of P_θ. This leads to a more general observation: induction on proof trees derived from a 'stripped' TSS is a powerful proof tool for TSS's with negative premises.

References

[1] J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. Syntax and defining equations for an interrupt mechanism in process algebra. *Fund. Inf.*, XI(2):127–168, 1986.

[2] B. Bloom, S. Istrail, and A.R. Meyer. Bisimulation can't be traced: preliminary report. In *Conference Record of the 15^{th} ACM Symposium on Principles of Programming Languages,* San Diego, California, pages 229–239, 1988.

[3] R.N. Bol and J.F. Groote. The meaning of negative premises in transition system specifications. Report CS-R9054, CWI, Amsterdam, 1990.

[4] T. Bolognesi, F. Lucidi, and S. Trigila. From timed Petri nets to timed LOTOS. In L. Logrippo, R.L. Probert, and H. Ural, editors, *Proceedings of the Tenth International IFIP WG6.1 Symposium on Protocol Specification, Testing and Verification,* Ottawa, 1990.

[5] J. Camilleri. An operational semantics for OCCAM. *International Journal of Parallel Programming*, 18(5):149–167, October 1989.

[6] R. Cleaveland and M. Hennessy. Priorities in process algebra. In *Proceedings third Annual Symposium on Logic in Computer Science* (LICS), Edinburgh, pages 193–202, 1988.

[7] A. van Gelder, K. Ross, and J.S. Schlipf. Unfounded sets and well-founded semantics for general logic programs. In *Proceedings of the Seventh Symposium on Principles of Database Systems*, pages 221–230. ACM SIGACT-SIGMOD, 1988.

[8] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. Kowalski and K. Bowen, editors, *Proceedings of the Fifth Logic Programming Symposium*, pages 1070–1080. MIT press, 1988.

[9] R.J. van Glabbeek. Bounded nondeterminism and the approximation induction principle in process algebra. In F.J. Brandenburg, G. Vidal-Naquet, and M. Wirsing, editors, *Proceedings STACS 87,* LNCS 247, pages 336–347. Springer-Verlag, 1987.

[10] J.F. Groote. Transition system specifications with negative premises. Technical Report CS-R8950, CWI, Amsterdam, 1989. An extended abstract appeared in J.C.M. Baeten and J.W. Klop, editors, *Proceedings of Concur90*, Amsterdam, LNCS 458, pages 332-341. Springer-Verlag, 1990.

[11] J.F. Groote and F.W. Vaandrager. Structured operational semantics and bisimulation as a congruence. Technical Report CS-R8845, CWI, Amsterdam, 1988. An extended abstract appeared in G. Ausiello, M. Dezani-Ciancaglini and, S. Ronchi Della Rocca, editors, *Proceedings ICALP 89*, Stresa, LNCS 372, pages 423–438. Springer-Verlag, 1989.

[12] R. Milner. *A Calculus of Communicating Systems.* LNCS 92. Springer-Verlag, 1980.

[13] D.M.R. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *Proceedings Fifth GI Conference,* LNCS 104, pages 167–183. Springer-Verlag, 1981.

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

[15] A. Pnueli. Linear and branching structures in the semantics and logics of reactive systems. In W. Brauer, editor, *Proceedings ICALP 85,* Nafplion, LNCS 194, pages 15–32. Springer-Verlag, 1985.

[16] T.C. Przymusinski. On the declarative semantics of deductive databases and logic programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 193–216. Morgan Kaufmann Publishers Inc., Los Altos, California, 1987.

[17] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.

[18] F.W. Vaandrager. *Algebraic Techniques for Concurrency and their Application.* PhD thesis, Centrum voor Wiskunde en Informatica, February 1990.