Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

R.N. Bol, J.F. Groote

The meaning of negative premises in transition system specifications

Computer Science/Department of Software Technology Report CS-R9054 October

The Meaning of Negative Premises in Transition System Specifications

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 [13, 12] to which TSS's are close. In an appendix we provide an extensive comparison between them.
As in [16], 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. We also extend the conservativity theorems of [16, 17] considerably. 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 [6, 16]. Finally, we provide a sound and complete axiomatization for this example.

Key Words & Phrases: Structured Operational Semantics (SOS), Transition System Specification (TSS), logic programming, negative premises, stable model semantics, well-founded model semantics, stratification, transition relation, bisimulation, congruence, *ntyft/ntyxt*-format, sum of TSS's, conservative extension of TSS's, priorities and abstraction, process algebra.
1985 Mathematics Subject Classification: 68Q60, 68Q55.
1982 CR Categories: D.3.1, F.3.1, F.3.2, I.2.3.
Note: 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 [18, 27] 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*) [17]. Recently, the use of negative premises in these rules has become popular [1, 6, 8, 16, 19, 20, 21, 22, 25, 28]. However, the logical meaning of those negative premises is not always clear. Therefore, the formal foundation of some of these articles is somewhat questionable. The problematic nature of negative premises has already been observed in [1, 6, 16].

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 [4, 32] and logic programming [13]. Without negative premises the notion of proof is standard. With negative premises we may only use the rules of which the negative premises hold. A negative premise holds by default, that is unless the opposite can be proved. Now suppose $\longrightarrow$ contains all transitions that can be proved in this way. Then it must satisfy:

> $\longrightarrow$ is the set of transitions that are provable by those rules of which the negative premises are consistent with $\longrightarrow$.

Following [13], we call such a transition relation ***stable*** for the TSS.

It is possible that a TSS has zero, one or more stable transition relations. If a TSS P has exactly one stable transition relation then we say that this relation is *associated with* P. If a TSS has zero or more than one stable transition relations, it is hard to imagine that any specific transition relation can be associated with it on reasonable grounds. That is, unless one is prepared to associate with a TSS a transition relation that is not decisive about all transitions. But this is not considered appropriate for the field of operational semantics.

In general it is difficult to show that a TSS has a unique stable transition relation. However, some techniques have been developed for finding a transition relation that is associated with a TSS.

The first technique, called ***stratification***, is presented in [16]. It is based on the notion of local stratification in logic programming [30]. In this article we show that the transition relation associated with a stratified TSS according to [16] is indeed the unique stable transition relation for it. This also implies the same fact for positive TSS's and TSS's in the so-called GSOS-format [6], as they are stratified.

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 [12, 31]. 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.

A desirable property for a TSS is that the strong bisimulation equivalence induced by it [24, 26] is a congruence. In [17] the *tyft/tyxt*-format was introduced, as a syntactical condition on TSS's that guarantees this property for positive TSS's. In [16] this condition was generalized to the *ntyft/ntyxt*-format for stratified TSS's. Here we show that the same condition is sufficient for all TSS's for which the reduction technique works. In contrast we show that the condition is not sufficient for TSS's having an associated transition relation that is not produced by reduction.

It can be useful to enrich a given language with additional language constructs. In order to do this in a systematic way, the *sum* of two TSS's has been introduced [17]. The sum of two TSS's P_0 and P_1 is called a conservative extension of P_0 if certain relevant properties of terms over the signature of P_0 are preserved. In [16] syntactical conditions on stratified TSS's were given ensuring that their sum is conservative. Here we generalize these conditions considerably and we extend them to TSS's for which the reduction technique works.

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 [2] and abstraction [14, 24]. We show using reduction and stratification that this TSS is meaningful. In section 10 we give a sound and complete axiomatization of strong bisimulation equivalence induced by this TSS. It turns out that most of the standard techniques for positive TSS's can still be used.

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

This paper is structured as follows:

2 Preliminaries

In this section we provide the basic concepts of this paper: *transition relations* and *Transition System Specifications (TSS's)*. An example of a TSS is given in which priorities and abstraction are integrated in BPA. This example will serve as a running example throughout this article.

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,
- $rank : F \rightarrow \mathbb{N}$ is a *rank function* which gives the arity of a function name; if $f \in F$ and $rank(f) = 0$ then f is called a *constant name.*

Let $W \subseteq V$ be a set of variables. The set of Σ-*terms* over W, notation $T(\Sigma, W)$, is the least set satisfying:

- $W \subseteq T(\Sigma, W)$,
- if $f \in F$ and $t_1, ..., t_{rank(f)} \in T(\Sigma, W)$, then $f(t_1, ..., t_{rank(f)}) \in T(\Sigma, W)$.

$T(\Sigma, \emptyset)$ 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.* Clearly, $T(\Sigma) \subset \mathbb{T}(\Sigma)$. $Var(t) \subseteq V$ is the set of variables in a term $t \in \mathbb{T}(\Sigma)$. A *substitution* σ is a mapping in $V \rightarrow \mathbb{T}(\Sigma)$. A substitution σ is extended to a mapping $\sigma : \mathbb{T}(\Sigma) \rightarrow \mathbb{T}(\Sigma)$ in a standard way by the following definition:

- $\sigma(f(t_1, ..., t_{rank(f)})) \;=\; f(\sigma(t_1), ..., \sigma(t_{rank(f)}))$ for $f \in F$ and $t_1, ..., t_{rank(f)} \in \mathbb{T}(\Sigma)$.

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 [18, 27] 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 [17] for a system in which rules had only positive premises. Negative premises were added in [16].

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:

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

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.* $t \overset{a}{\not\longrightarrow}$ is called a *negative literal.* φ, ψ, χ are used to range over literals. For a literal ψ, *source*(ψ) denotes the term at the left hand side of ψ and, if ψ is positive, *target*(ψ) denotes the term at the right hand side. 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 [2, 3, 9, 10]. We follow the line set out by BAETEN, BERGSTRA and KLOP [2] who introduced a priority operator θ. For abstraction we take observation equivalence as introduced by MILNER [24] although technically we follow VAN GLABBEEK [14]. We base our example on BPA$_{\delta\epsilon\tau}$, *Basic Process Algebra* with τ, ϵ and δ as introduced in [17], and extend it with recursion and priorities.

Example 2.4 (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 two special constants δ and ϵ. δ is called *inaction* (or *deadlock*) and it represents the process that cannot do anything at all. In particular, δ cannot terminate. ϵ is called the *empty process* which cannot do anything but terminate.

$\epsilon:$	R1:	$\epsilon\xrightarrow{\surd}\delta$		
$a:$	R2:	$a\xrightarrow{a}\epsilon$ if $a\in Act_\tau$		
$+:$	R3.1:	$\dfrac{x\xrightarrow{a}x'}{x+y\xrightarrow{a}x'}$	R3.2:	$\dfrac{y\xrightarrow{a}y'}{x+y\xrightarrow{a}y'}$
$\cdot:$	R4.1:	$\dfrac{x\xrightarrow{a}x'}{x\cdot y\xrightarrow{a}x'\cdot y}$ if $a\in Act_\tau$	R4.2:	$\dfrac{x\xrightarrow{\surd}x' \quad y\xrightarrow{a}y'}{x\cdot y\xrightarrow{a}y'}$
$\theta:$	R5.1:	$\dfrac{x\xrightarrow{a}x' \quad \forall b>a\ x\not\xrightarrow{b}}{\theta(x)\xrightarrow{a}\theta(x')}$ if $a\in Act_\tau$	R5.2:	$\dfrac{x\xrightarrow{\surd}x'}{\theta(x)\xrightarrow{\surd}\theta(x')}$
$\triangleleft:$	R6.1:	$\dfrac{x\xrightarrow{a}x' \quad \forall b>a\ y\not\xrightarrow{b}}{x\triangleleft y\xrightarrow{a}x'}$ if $a\in Act_\tau$	R6.2:	$\dfrac{x\xrightarrow{\surd}x'}{x\triangleleft y\xrightarrow{\surd}x'}$
$\tau_I:$	R7.1:	$\dfrac{x\xrightarrow{a}x'}{\tau_I(x)\xrightarrow{a}\tau_I(x')}$ if $a\notin I$	R7.2:	$\dfrac{x\xrightarrow{a}x'}{\tau_I(x)\xrightarrow{\tau}\tau_I(x')}$ if $a\in I$
recursion:	R8:	$\dfrac{t_X\xrightarrow{a}y}{X\xrightarrow{a}y}$ if $X\Leftarrow t_X\in E$		
τ-rules:	R9.1:	$a\xrightarrow{a}\tau$ if $a\in Act_\tau$		
	R9.2:	$\dfrac{x\xrightarrow{\tau}y \quad y\xrightarrow{a}z}{x\xrightarrow{a}z}$	R9.3:	$\dfrac{x\xrightarrow{a}y \quad y\xrightarrow{\tau}z}{x\xrightarrow{a}z}$

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

Two basic operators compose smaller into larger processes: *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 abstracted away: for all $I \subseteq Act$ the unary *abstraction operator* τ_I performs this task by renaming all actions in I to τ.

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 BPA$_{\delta\epsilon\tau}$ with priorities, which is given in section 10, we need the *unless* operator $\triangleleft$, which was introduced in [2]. 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 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 $\surd$ which is used to signal termination. 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 $\forall b > a \; x \overset{b}{\not\longrightarrow}$ which means that for all b with higher priority than a, there is a negative premise $x \overset{b}{\not\longrightarrow}$. Rule R5.1 is intuitively appealing. It says that $\theta(x)$ may do an a-action if x can do this action and x cannot do any action with higher priority. But there is a snag in it. Due to the negative premises, it is not at all straightforward to see that P_θ defines a transition relation. In fact, in example 4.8 we will present a case in which it does not make sense at all.

Rules R9.1-R9.3 model the properties of τ. R9.2 and R9.3 say that whenever an action a is observed in some time interval, numerous unobservable τ-actions can also happen during the same time, both before and after a. Rule R9.1 says that if an action a is observed, some internal activity may exist before the next action can take place.

We think that the remaining rules are self explanatory, although we like to point out that rule R4.2 makes use of a process that explicitly signals termination.

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 that these techniques are not capable of handling our running example satisfactorily. In the next sections we show how to solve this problem.

As a first step, 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 literal ψ *holds* in $\longrightarrow$ or ψ is *valid* in $\longrightarrow$, notation $\longrightarrow \models \psi$, if the transition $\psi \in \longrightarrow$. A negative literal $t \overset{a}{\not\longrightarrow}$ *holds* in $\longrightarrow$, notation $\longrightarrow \models t \overset{a}{\not\longrightarrow}$, if for no $t' \in T(\Sigma)$ the transition $t \xrightarrow{a} t' \in \longrightarrow$. For a set of literals Ψ, we write $\longrightarrow \models \Psi$ iff $\forall \psi \in \Psi$: $\longrightarrow \models \psi$.

Remark 3.2. Suppose we have two transition relations $\longrightarrow_1$ and $\longrightarrow_2$ such that $\longrightarrow_1 \subseteq \longrightarrow_2$. For any set of positive literals Ψ it is clear that $\longrightarrow_1 \models \Psi$ implies $\longrightarrow_2 \models \Psi$. However, if Ψ is a set of negative literals, then $\longrightarrow_2 \models \Psi$ implies $\longrightarrow_1 \models \Psi$. We shall often use this kind of reasoning.

What is the transition relation defined by a TSS? 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 3.3. 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 3.4. 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 3.5. 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 [6] 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. This is shown by the following example.

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

$$\frac{f \xrightarrow{a} f}{f \xrightarrow{a} f}.$$

We would like P to define the transition relation $\longrightarrow_P = \emptyset$. We feel that there is not enough reason to add $f \xrightarrow{a} f$ to $\longrightarrow_P$ as it can only be 'derived' by assuming that it is already in $\longrightarrow_P$.
However, both $\emptyset$ and $\{f \xrightarrow{a} f\}$ are supported models of P.

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. The existence of this least model follows from the model intersection property stated below.

Lemma 3.7 *(Model intersection property). Let P be a TSS and let $\mathcal{C}$ be a collection of models of P. Then $\bigcap \mathcal{C}$ is a model of P.*
Proof. Let r be a ground instance of a rule of P. If $\bigcap \mathcal{C} \models prem(r)$, then for every $\longrightarrow \in \mathcal{C}$: $\longrightarrow \models prem(r)$, thus for every $\longrightarrow \in \mathcal{C}$: $\longrightarrow \models conc(r)$, as $\mathcal{C}$ is a collection of models. Thus $\bigcap \mathcal{C} \models conc(r)$. □

Thus we have the following definition.

Definition 3.8. The transition relation $\longrightarrow_P$ *associated with* a positive TSS P is the least model of P w.r.t. set inclusion.

Traditionally ([17, 18, 27]) a different definition of the transition relation associated with a positive TSS was given, based on the *provability* of transitions. We show that these two characterizations are equivalent.

Definition 3.9. Let $P = (\Sigma, A, R)$ be a positive TSS. A *proof* of a positive literal ψ from P is a well-founded, upwardly branching tree of which the nodes are labeled by literals $t \xrightarrow{a} t'$ with $t, t' \in \mathbb{T}(\Sigma)$ and $a \in A$, such that:

- the root is labeled with ψ,
- if χ is the label of a node q and $\{\chi_i | i \in I\}$ is the set of labels of the nodes directly above q, then there is a rule $\dfrac{\{\varphi_i | i \in I\}}{\varphi}$ in R and a substitution $\sigma: V \to \mathbb{T}(\Sigma)$ such that $\chi = \sigma(\varphi)$ and $\chi_i = \sigma(\varphi_i)$ for $i \in I$.

A proof is *closed* if it only contains closed literals. A positive literal ψ is *provable* in P, notation $P \vdash \psi$, if there exists a proof of ψ from P.

Theorem 3.10. *Let $P = (\Sigma, A, R)$ be a positive TSS, $\longrightarrow_P$ the transition relation associated with P and $\psi \in Tr(\Sigma, A)$. Then*

$$P \vdash \psi \quad \Leftrightarrow \quad \psi \in \longrightarrow_P.$$

Proof.

$\Rightarrow$ By straightforward induction on the proof of ψ from P.

$\Leftarrow$ It is straightforward to show that $\{\psi | P \vdash \psi\}$ is a model of P. As $\longrightarrow_P$ is the least model of P, it follows that $\longrightarrow_P \subseteq \{\psi | P \vdash \psi\}$.

□

From this theorem it also follows that the least model of a positive TSS is supported by it.

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

$$\frac{f \xrightarrow{a} f}{f \xrightarrow{a} f} \qquad \frac{f \overset{a}{\nrightarrow}}{f \xrightarrow{b} f}.$$

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. The correspondence between TSS's and logic programs is treated in appendix A. A first solution proposed there introduced the notion of (local) stratification. Here we follow [16], where this notion was tailored for TSS's.

A TSS P is stratified if there exists a *stratification* of the transitions with respect to the rules of P. The stratification guarantees that for no literal its validity depends negatively on itself.

Definition 3.12. Let $P = (\Sigma, A, R)$ be a TSS. A function $S : Tr(\Sigma, A) \rightarrow \alpha$, where α is an ordinal, is called a *stratification* of P if for every rule $r \in R$ and every substitution $\sigma : V \rightarrow T(\Sigma)$ it holds that:

$$\begin{aligned}&\text{for all } \psi \in pprem(\sigma(r)) : \ S(\psi) \leq S(conc(\sigma(r)) \text{ and}\\ &\text{for all } t \overset{a}{\not\longrightarrow} \in nprem(\sigma(r)) \text{ and } t' \in T(\Sigma) : \ S(t \xrightarrow{a} t') < S(conc(\sigma(r))).\end{aligned}$$

If P has a stratification, we say that P is *stratified.* For all ordinals $\beta < \alpha$, $S_\beta = \{\varphi | S(\varphi) = \beta\}$ is called a *stratum.*

Example 3.13. The TSS of example 3.11 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 [16] it is shown that $\text{BPA}_{\epsilon\delta}$ with priorities and renaming but without abstraction is stratified under some appropriate conditions.

We now define how a transition relation $\longrightarrow_{P,S}$ is constructed from a TSS P with stratification S, rephrasing a corresponding definition in [16]. The idea of the construction is that one first considers the positive literals in stratum 0. As each literal in stratum 0 can only fit the conclusion of a rule without negative premises, one can determine which of these literals hold and which do not hold in $\longrightarrow_{P,S}$ in the same way as is done for positive transition system specifications. If a literal in stratum 1 fits the conclusion of a rule, then this instance of that rule can only have negative premises in stratum 0. If these negative premises hold (which has already been determined), they can be discarded. If they do not hold, the rule cannot be applied. Then we can prove the literals in stratum 1 in the ordinary way and we proceed with stratum 2 etc.

Definition 3.14. Let $P = (\Sigma, A, R)$ be a TSS with a stratification $S : Tr(\Sigma, A) \rightarrow \alpha$ for some ordinal α. The transition relation $\longrightarrow_{P,S}$ *associated with* P (and based on S) is defined as:

$$\longrightarrow_{P,S} = \bigcup_{0 \leq i < \alpha} \longrightarrow_{P_i}.$$

where $\longrightarrow_{P_i}$ is defined by the (positive) TSS $P_i = (\Sigma, A, R_i)$ with R_i given by:

$$\begin{aligned}R_i = \{r' | \exists r \in R \text{ and } \exists \sigma : V \rightarrow T(\Sigma) :\\ &\bigcup_{0 \leq j < i} \longrightarrow_{P_j} \models nprem(\sigma(r)) \cup \{\varphi \in pprem(\sigma(r)) | S(\varphi) < i\},\\ &S(conc(\sigma(r))) = i \text{ and}\\ &r' = \frac{\{\varphi \in pprem(\sigma(r)) | S(\varphi) = i\}}{conc(\sigma(r))}\}.\end{aligned}$$

Theorem 3.15 *(See lemma 2.5.4 in [16]). Let P be a TSS which is stratified by stratifications S and S'. Then $\longrightarrow_{P,S} = \longrightarrow_{P,S'}$.*

This theorem allows us to write $\longrightarrow_P$ for the transition relation associated with a stratified TSS P. Note that the definition of $\longrightarrow_P$ based on the notion of 'stratification' extends the definition of $\longrightarrow_P$ for positive TSS's.

Theorem 3.16 *(See theorem 2.5.3 in [16] and theorem 4 in [30]). Let P be a stratified TSS. Then $\longrightarrow_P$ is a minimal and supported model of P.*

$$\begin{tikzcd}[column sep=small]
 & & \cdot\ \text{Any transition relation} & & & \\
 & \text{Supported}\ \cdot \arrow[ur] & & \cdot\ \text{Model} \arrow[ul] & & \\
 & & \text{Supported model}\ \cdot \arrow[ul] \arrow[ur] & & *\ \text{Minimal model} \arrow[ul] & \\
 & & & \text{Supported and minimal model}\ * \arrow[ul] \arrow[ur] & & *\ \text{Least model} \arrow[ul] \\
 & & & & *\ \text{Model based on stratification} \arrow[ul] &
\end{tikzcd}$$

Figure 1: Relations among several models

Thus we have the scheme of characterizations depicted in figure 1, where $A \longrightarrow B$ means that characterization A implies characterization B. For positive TSS's, the characterizations marked by a $*$ coincide.

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 BPA$_{\delta\epsilon\tau}$ with priorities.

Example 3.17. Suppose we have an instance P_θ of 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:

$$\text{R5.1:}\quad \frac{t \xrightarrow{a} u \quad t \overset{b}{\not\longrightarrow}}{\theta(t) \xrightarrow{a} \theta(u)},$$

$$\text{R7.2:}\quad \frac{\theta(t) \xrightarrow{a} \theta(u)}{\tau_{\{a\}}(\theta(t)) \xrightarrow{\tau} \tau_{\{a\}}(\theta(u))},$$

$$\text{R9.3:}\quad \frac{t \xrightarrow{b} \tau_{\{a\}}(\theta(t)) \quad \tau_{\{a\}}(\theta(t)) \xrightarrow{\tau} \tau_{\{a\}}(\theta(u))}{t \xrightarrow{b} \tau_{\{a\}}(\theta(u))}.$$

For any stratification S of P_θ it should thus hold that

$$\begin{aligned}
&S(t \xrightarrow{b} \tau_{\{a\}}(\theta(u))) < && \text{(R5.1)}\\
&S(\theta(t) \xrightarrow{a} \theta(u)) \leq && \text{(R7.2)}\\
&S(\tau_{\{a\}}(\theta(t)) \xrightarrow{\tau} \tau_{\{a\}}(\theta(u))) \leq && \text{(R9.3)}\\
&S(t \xrightarrow{b} \tau_{\{a\}}(\theta(u))).
\end{aligned}$$

Of course, such a stratification cannot exist.

Again, this problem has been recognized earlier in logic programming, and several more powerful techniques were introduced there [29, 5, 13, 12]. In the following two sections we adapt [13] and [12] for TSS's.

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 [13]. 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, $t \overset{a}{\not\longrightarrow}$ 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 $\longrightarrow \subseteq Tr(\Sigma, A)$.

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

where

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

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 $\mathit{Strip}(P, \longrightarrow)$ 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 $\longrightarrow = \longrightarrow_{\mathit{Strip}(P, \longrightarrow)}$.

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.

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

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

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

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.

The following lemma will be used implicitly in almost every proof to follow. Moreover, it shows that 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.

Lemma 4.7. *Let $P = (\Sigma, A, R)$ be a TSS and let $\longrightarrow \subseteq Tr(\Sigma, A)$ be a transition relation. If $\longrightarrow$ is stable for P, then*

1. *$\longrightarrow$ is a model of P,*
2. *$\longrightarrow$ is supported by P,*
3. *$\longrightarrow$ is a minimal model of P (Cf. [13], theorem 1).*

Proof. Let $\longrightarrow$ be a transition relation that is stable for P.

1. Suppose $r \in R$ and $\longrightarrow \models prem(r)$. Hence
$$\frac{pprem(r)}{conc(r)} \in Strip(R, \longrightarrow).$$
As $\longrightarrow = \longrightarrow_{Strip(P,\longrightarrow)}$ is a model of $Strip(P, \longrightarrow)$ and $\longrightarrow \models pprem(r)$, $\longrightarrow \models conc(r)$.

2. Suppose $\varphi \in \longrightarrow$. Hence, $\longrightarrow \models \varphi$ and thus $\longrightarrow_{Strip(P,\longrightarrow)} \models \varphi$. This means that there is a proof for φ using the rules in $Strip(R, \longrightarrow)$. Assume rule r is the last rule used. So $conc(r) = \varphi$. Hence $Strip(P, \longrightarrow) \vdash prem(r)$ and thus $\longrightarrow \models prem(r)$. As $r \in Strip(R, \longrightarrow)$ there is a rule
$$r' = \frac{prem(r) \cup nprem}{conc(r)} \in R$$

where *nprem* is a set of negative premises such that $\longrightarrow \models \mathit{nprem}$. Hence, for this rule $r' \in R$ it holds that $\varphi = \mathit{conc}(r')$ and $\longrightarrow \models \mathit{prem}(r')$. Hence $\longrightarrow$ is supported by P.

3. We must show that $\longrightarrow$ is minimal among the models of P. Suppose $\longrightarrow^* \subseteq \longrightarrow$ is also a model of P. We show that $\longrightarrow^*$ is a model of $\mathit{Strip}(P, \longrightarrow)$. Let r be a rule of $\mathit{Strip}(P, \longrightarrow)$. This means that there is some
$$r' = \frac{\mathit{prem}(r) \cup \mathit{nprem}}{\mathit{conc}(r)} \in R$$
for some set *nprem* of negative premises. As $\longrightarrow^* \subseteq \longrightarrow$ and $\longrightarrow \models \mathit{nprem}$, $\longrightarrow^* \models \mathit{nprem}$. As $\longrightarrow^*$ is a model of P, we have

   if $\longrightarrow^* \models \mathit{prem}(r) \cup \mathit{nprem}$, then $\longrightarrow^* \models \mathit{conc}(r)$.

   Knowing that $\longrightarrow^* \models \mathit{nprem}$, this reduces to

   if $\longrightarrow^* \models \mathit{prem}(r)$, then $\longrightarrow^* \models \mathit{conc}(r)$.

   Thus $\longrightarrow^*$ is a model for every rule r in $\mathit{Strip}(P, \longrightarrow)$. As $\longrightarrow$ is the least model of $\mathit{Strip}(P, \longrightarrow)$, it follows that $\longrightarrow^* = \longrightarrow$.

□

We show how the notion *is stable for* can be applied to our running example. What we in fact show is that in general there is no stable transition relation for $\mathrm{BPA}_{\delta\epsilon\tau}$ with priorities instantiated with a set of process declarations where the abstraction operator τ_I is allowed in process terms.

Example 4.8. Consider P_θ with at least two actions a and b such that $a > b$ and a process name X with the recursive definition
$$X \Leftarrow \theta(\tau_{\{b\}}(X) \cdot a + b) \in E.$$
Now assume that there is a relation $\longrightarrow$ that is stable for P_θ. We show that this assumption leads to a contradiction. For a more convenient notation, we use $t\xrightarrow{a}$ as an abbreviation of $\exists u \in T(\Sigma_\theta) : \longrightarrow \models t\xrightarrow{a}u$ ($t \in T(\Sigma_\theta)$, $a \in A_\theta$). We distinguish three cases but we do not present them in full detail. In particular not all possible applications of R9.2 and R9.3 are considered explicitly.

- $\tau_{\{b\}}(X)\xrightarrow{\surd}$. As $\longrightarrow$ is a model of P_θ, we have that $\tau_{\{b\}}(X) \cdot a\xrightarrow{a}$ (by rule R4.2) and hence $\tau_{\{b\}}(X) \cdot a + b\xrightarrow{a}$ (by rule R3.1). Thus $\theta(\tau_{\{b\}}(X) \cdot a + b)\xrightarrow{a}$ and $\longrightarrow \models \theta(\tau_{\{b\}}(X) \cdot a + b)\not\xrightarrow{b}$ (by rule R5.1). So $\longrightarrow \models X\not\xrightarrow{b}$. As obviously $\longrightarrow \models X\not\xrightarrow{\surd}$ (X must perform at least an a or b action), it follows that $\longrightarrow \models \tau_{\{b\}}(X)\not\xrightarrow{\surd}$. Contradiction.
- $\longrightarrow \models \tau_{\{b\}}(X)\not\xrightarrow{\surd}$ and $\longrightarrow \models \tau_{\{b\}}(X)\not\xrightarrow{a}$. Then obviously $\longrightarrow \models \tau_{\{b\}}(X) \cdot a + b\not\xrightarrow{a}$, so $\longrightarrow \models \theta(\tau_{\{b\}}(X) \cdot a + b)\xrightarrow{b}\theta(\epsilon)$ (using R2, R3.2 and R5.1). Hence, $\longrightarrow \models X\xrightarrow{b}\theta(\epsilon)$, so $\longrightarrow \models \tau_{\{b\}}(X)\xrightarrow{\surd}\tau_{\{b\}}(\theta(\delta))$ (using R8, R7.2, R1, R5.2, R7.2 and R9.2). Contradiction.
- $\longrightarrow \models \tau_{\{b\}}(X)\not\xrightarrow{\surd}$ and $\tau_{\{b\}}(X)\xrightarrow{a}t$ for some $t \in T(\Sigma_\theta)$. This can also not be the case as there is no proof for $\mathit{Strip}(P_\theta, \longrightarrow) \vdash \tau_{\{b\}}(X)\xrightarrow{a}t$. In order to prove $\tau_{\{b\}}(X)\xrightarrow{a}t$, we must show that $X\xrightarrow{a}$, in order to show this we need $\tau_{\{b\}}(X) \cdot a\xrightarrow{a}$. In combination with the assumption that $\tau_{\{b\}}(X)\not\xrightarrow{\surd}$, this requires $\tau_{\{b\}}(X)\xrightarrow{a}$ again. Thus the most 'reasonable' attempt to construct the required proof loops. All other attempts to prove $\tau_{\{b\}}(X)\xrightarrow{a}t$, e.g. via R9.3:
$$\frac{\tau_{\{b\}}(X)\xrightarrow{a}u \quad u\xrightarrow{\tau}t}{\tau_{\{b\}}(X)\xrightarrow{a}t},$$
also loop.

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 [12]. 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 $t \xrightarrow{a}\!\!\!\!/\;$ 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 $t \xrightarrow{a}\!\!\!\!/\;$ 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 $t \xrightarrow{a}\!\!\!\!/\;$ is certainly true if t cannot possibly perform an a-step, i.e. for no t': $t \xrightarrow{a} t'$ is possible. Thus $t \xrightarrow{a}\!\!\!\!/\;$ is certainly true if $\longrightarrow_{pos} \models t \xrightarrow{a}\!\!\!\!/\;$.

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 $Pos(R) = \{r' | \exists r \in R :\ r' = \dfrac{pprem(r)}{conc(r)}\}$.

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 [12] and general practice in logic programming, our operator maps TSS's to TSS's rather than interpretations to interpretations; for details see appendix A. This allows us in section 6 to combine reduction with stratification: as soon as the reduced TSS is stratified, no further reduction is needed.

The following lemma plays an important role in a number of proofs to follow. It shows that the reduction process can never make a certainly true (or false) literal become unknown. Thus reduction is monotonic in this sense.

Lemma 5.5 *(Monotonicity of reduction). Let $P = (\Sigma, A, R)$ be a TSS. For all ordinals β and α such that $\beta < \alpha$ and for every $\longrightarrow \subseteq Tr(\Sigma, A)$:*

$$\begin{aligned} \longrightarrow_{True(Red^{\beta}(P))} \subseteq \longrightarrow_{True(Red^{\alpha}(P))} &\subseteq \\ \longrightarrow_{Strip(Red^{\alpha}(P),\longrightarrow)} &\subseteq \\ \longrightarrow_{Pos(Red^{\alpha}(P))} &\subseteq \longrightarrow_{Pos(Red^{\beta}(P))}. \end{aligned}$$

Proof.

(1) First we show that $\longrightarrow_{True(Red^{\alpha}(P))} \subseteq \longrightarrow_{Strip(Red^{\alpha}(P),\longrightarrow)} \subseteq \longrightarrow_{Pos(Red^{\alpha}(P))}$. For every TSS $P' = (\Sigma, A, R')$: $True(R') \subseteq Strip(R', \longrightarrow) \subseteq Pos(R')$. As these TSS's are all positive, $\longrightarrow_{True(P')} \subseteq \longrightarrow_{Strip(P',\longrightarrow)} \subseteq \longrightarrow_{Pos(P')}$. Now taking $P' = Red^{\alpha}(P)$ proves this case.

(2) Here it is shown that $\longrightarrow_{Pos(Red^{\alpha}(P))} \subseteq \longrightarrow_{Pos(Red^{\beta}(P))}$. Suppose $Pos(Red^{\alpha}(P)) \vdash \varphi$. Then there is a rule $r' \in Pos(Red^{\alpha}(P))$ such that $conc(r') = \varphi$ and $Pos(Red^{\alpha}(P)) \vdash prem(r')$. Hence, there is a rule $r \in R$ such that $conc(r) = \varphi$ and

$$\bigcup_{\gamma<\alpha} \longrightarrow_{True(Red^{\gamma}(P))} \models nprem(r),$$
$$\bigcap_{\gamma<\alpha} \longrightarrow_{Pos(Red^{\gamma}(P))} \models pprem(r).$$

Now

$$\bigcup_{\gamma<\beta} \longrightarrow_{True(Red^{\gamma}(P))} \subseteq \bigcup_{\gamma<\alpha} \longrightarrow_{True(Red^{\gamma}(P))},$$

$$\bigcap_{\gamma<\beta} \longrightarrow_{Pos(Red^{\gamma}(P))} \supseteq \bigcap_{\gamma<\alpha} \longrightarrow_{Pos(Red^{\gamma}(P))}.$$

Hence, $\bigcup_{\gamma<\alpha} \longrightarrow_{True(Red^{\gamma}(P))} \models nprem(r)$ implies that $\bigcup_{\gamma<\beta} \longrightarrow_{True(Red^{\gamma}(P))} \models nprem(r)$. Conversely, $\bigcap_{\gamma<\alpha} \longrightarrow_{Pos(Red^{\gamma}(P))} \models pprem(r)$ implies $\bigcap_{\gamma<\beta} \longrightarrow_{Pos(Red^{\gamma}(P))} \models pprem(r)$. Hence $Pos(Red^{\beta}(R))$ contains a rule r'' such that $prem(r'') \subseteq pprem(r)$ and $conc(r'') = \varphi$. As also $\bigcap_{\gamma<\alpha} \longrightarrow_{Pos(Red^{\gamma}(P))} \models pprem(r)$ implies $\longrightarrow_{Pos(Red^{\beta}(P))} \models pprem(r)$, it holds that $\longrightarrow_{Pos(Red^{\beta}(P))} \models prem(r'')$. Thus $Pos(Red^{\beta}(P)) \vdash \varphi$.

(3) We are left to show $\longrightarrow_{True(Red^{\beta}(P))} \subseteq \longrightarrow_{True(Red^{\alpha}(P))}$. We show this by induction on α. By induction we may assume that:

$$\text{for all } \zeta \leq \gamma < \alpha: \quad \longrightarrow_{True(Red^{\zeta}(P))} \subseteq \longrightarrow_{True(Red^{\gamma}(P))}.$$

Suppose $True(Red^{\beta}(P)) \vdash \varphi$. Then there is an instance r of a rule in R such that $conc(r) = \varphi$,

$$\bigcap_{\gamma<\beta} \longrightarrow_{Pos(Red^{\gamma}(P))} \models nprem(r)$$

(all negative premises of r must be removed by reduction) and

$$\bigcup_{\gamma\leq\beta} \longrightarrow_{True(Red^{\gamma}(P))} \models pprem(r)$$

(all positive premises of r must be removed by reduction or proved in $True(Red^{\beta}(P))$). From this, using the induction hypothesis (i.h.) and (1) and (2) above, we can infer the following two facts:

- $\bigcap_{\gamma<\beta} \longrightarrow_{Pos(Red^{\gamma}(P))} \models nprem(r) \Rightarrow$
  $\bigcap_{\gamma\leq\beta} \longrightarrow_{Pos(Red^{\gamma}(P))} \models nprem(r) \overset{(2)}{\Rightarrow}$
  $\longrightarrow_{Pos(Red^{\beta}(P))} \models nprem(r) \overset{(2)}{\Rightarrow}$
  $\forall\gamma\ (\beta \leq \gamma < \alpha)$: $\longrightarrow_{Pos(Red^{\gamma}(P))} \models nprem(r) \overset{(1)}{\Rightarrow}$
  $\forall\gamma\ (\beta \leq \gamma < \alpha)$: $\longrightarrow_{True(Red^{\gamma}(P))} \models nprem(r) \overset{(i.h.)}{\Rightarrow}$
  $\forall\gamma < \alpha$: $\longrightarrow_{True(Red^{\gamma}(P))} \models nprem(r) \Rightarrow$
  $\bigcup_{\gamma<\alpha} \longrightarrow_{True(Red^{\gamma}(P))} \models nprem(r)$. <1>
- $\bigcup_{\gamma\leq\beta} \longrightarrow_{True(Red^{\gamma}(P))} \models pprem(r) \overset{(i.h.)}{\Rightarrow}$
  $\longrightarrow_{True(Red^{\beta}(P))} \models pprem(r) \overset{(i.h.)}{\Rightarrow}$
  $\forall\gamma\ (\beta \leq \gamma < \alpha)$: $\longrightarrow_{True(Red^{\gamma}(P))} \models pprem(r) \overset{(1)}{\Rightarrow}$
  $\forall\gamma\ (\beta \leq \gamma < \alpha)$: $\longrightarrow_{Pos(Red^{\gamma}(P))} \models pprem(r) \overset{(2)}{\Rightarrow}$
  $\forall\gamma < \alpha$: $\longrightarrow_{Pos(Red^{\gamma}(P))} \models pprem(r) \Rightarrow$
  $\bigcap_{\gamma<\alpha} \longrightarrow_{Pos(Red^{\gamma}(P))} \models pprem(r)$. <2>

<1> and <2> imply $\exists r' = \dfrac{prem(r) - \ldots}{\varphi} \in Red^{\alpha}(R)$.
Furthermore,

$$\bigcap_{\gamma<\beta} \longrightarrow_{Pos(Red^{\gamma}(P))} \models nprem(r) \quad \Rightarrow$$

$$\bigcap_{\gamma<\alpha} \longrightarrow_{Pos(Red^{\gamma}(P))} \models nprem(r) \quad \Rightarrow$$

$$nprem(r') = \emptyset \quad \Rightarrow$$

$$r' \in True(Red^{\alpha}(R)).$$

As for every $\psi \in prem(r)$, the proof of ψ in $True(Red^{\beta}(P))$ is less deep than the proof of φ in $True(Red^{\beta}(P))$, we may conclude by induction that $prem(r) \subseteq \longrightarrow_{True(Red^{\alpha}(P))}$. As $prem(r') \subseteq prem(r)$, $conc(r') = \varphi \in \longrightarrow_{True(Red^{\alpha}(P))}$.

□

In order to apply this reduction process, we also need the following lemma.

Lemma 5.6. *Let $P = (\Sigma, A, R)$ be a TSS and let $\longrightarrow \subseteq Tr(\Sigma, A)$. For all ordinals α*

$$\bigcup_{\beta<\alpha} \longrightarrow_{True(Red^{\beta}(P))} \subseteq \longrightarrow \subseteq \bigcap_{\beta<\alpha} \longrightarrow_{Pos(Red^{\beta}(P))}$$

implies

$$\longrightarrow_{Strip(P,\longrightarrow)} = \longrightarrow_{Strip(Red^{\alpha}(P),\longrightarrow)}.$$

Proof.

- $\longrightarrow_{Strip(P,\longrightarrow)} \subseteq \longrightarrow_{Strip(Red^{\alpha}(P),\longrightarrow)}$. Let $\psi \in \longrightarrow_{Strip(P,\longrightarrow)}$. Hence, $Strip(P,\longrightarrow) \vdash \psi$. We use induction on this proof. There is a rule r' in $Strip(P,\longrightarrow)$ such that $conc(r') = \psi$ and $\longrightarrow_{Strip(P,\longrightarrow)} \models prem(r')$. Hence, there is a rule $r \in R$ such that

$$r' = \frac{pprem(r)}{conc(r)}$$

and $\longrightarrow \models nprem(r)$. By induction on α and lemma 5.5 it follows that

$$\forall \beta < \alpha : \longrightarrow_{Strip(P,\longrightarrow)} = \longrightarrow_{Strip(Red^{\beta}(P),\longrightarrow)} \subseteq \longrightarrow_{Pos(Red^{\beta}(P))}$$

and hence:

$$\left.\begin{array}{l} \longrightarrow \models nprem(r) \\ \bigcup_{\beta<\alpha} \longrightarrow_{True(Red^{\beta}(P))} \subseteq \longrightarrow \end{array}\right\} \Rightarrow \bigcup_{\beta<\alpha} \longrightarrow_{True(Red^{\beta}(P))} \models nprem(r),$$

$$\left.\begin{array}{l} \longrightarrow_{Strip(P,\longrightarrow)} \models pprem(r) \\ \longrightarrow_{Strip(P,\longrightarrow)} \subseteq \bigcap_{\beta<\alpha} \longrightarrow_{Pos(Red^{\beta}(P))} \end{array}\right\} \Rightarrow \bigcap_{\beta<\alpha} \longrightarrow_{Pos(Red^{\beta}(P))} \models pprem(r).$$

So there is a rule

$$r'' = \frac{\begin{array}{l}\{\psi' \in pprem(r) \mid \bigcup_{\beta<\alpha} \longrightarrow_{True(Red^{\beta}(P))} \not\models \psi'\} \cup \\ \qquad \{\psi' \in nprem(r) \mid \bigcap_{\beta<\alpha} \longrightarrow_{Pos(Red^{\beta}(P))} \not\models \psi'\}\end{array}}{conc(r)} \in Red^{\alpha}(P).$$

Furthermore, $\longrightarrow \models nprem(r'') \subseteq nprem(r)$. So

$$r''' = \frac{\{\psi' \in pprem(r) \mid \bigcup_{\beta<\alpha} \longrightarrow_{True(Red^{\beta}(P))} \not\models \psi'\}}{conc(r)} \in Strip(Red^{\alpha}(P),\longrightarrow).$$

By induction on the depth of the proof of ψ from $Strip(P,\longrightarrow)$ we may assume that $prem(r') \subseteq \longrightarrow_{Strip(Red^{\alpha}(P),\longrightarrow)}$, so $\longrightarrow_{Strip(Red^{\alpha}(P),\longrightarrow)} \models prem(r') = pprem(r) \supseteq prem(r''')$. Hence, it follows that $Strip(Red^{\alpha}(P),\longrightarrow) \models conc(r''') = \psi$.

- $\longrightarrow_{Strip(P,\longrightarrow)} \supseteq \longrightarrow_{Strip(Red^{\alpha}(P,\longrightarrow))}$. Let $\psi \in \longrightarrow_{Strip(Red^{\alpha}(P),\longrightarrow)}$. Hence, it must be that $Strip(Red^{\alpha}(P),\longrightarrow) \vdash \psi$. So there is a rule r' in $Strip(Red^{\alpha}(P),\longrightarrow)$ such that $conc(r') = \psi$ and $\longrightarrow_{Strip(Red^{\alpha}(P),\longrightarrow)} \models prem(r')$. Then there is a rule $r \in R$ such that

$$r' = \frac{\{\psi' \in pprem(r) \mid \bigcup_{\beta<\alpha} \longrightarrow_{True(Red^{\beta}(P))} \not\models \psi'\}}{conc(r)}$$

and $\longrightarrow \models \{\psi \in nprem(r) \mid \bigcap_{\beta<\alpha} \longrightarrow_{Pos(Red^{\beta}(P))} \not\models \psi'\}$.

$\longrightarrow \models nprem(r)$: Let $\psi' \in nprem(r)$. If $\bigcap_{\beta<\alpha} \longrightarrow_{Pos(Red^\beta(P))} \not\models \psi'$ then $\longrightarrow \models \psi'$. If $\bigcap_{\beta<\alpha} \longrightarrow_{Pos(Red^\beta(P))} \models \psi'$ then also $\longrightarrow \models \psi'$, as $\longrightarrow \subseteq \bigcap_{\beta<\alpha} \longrightarrow_{Pos(Red^\beta(P))}$.

So in $Strip(P, \longrightarrow)$ we have the rule

$$r'' = \frac{pprem(r)}{conc(r)}.$$

By induction on the depth of the proof of ψ from $Strip(Red^\alpha(P), \longrightarrow)$ we may conclude that $\longrightarrow_{Strip(P,\longrightarrow)} \models prem(r') = \{\psi' \in pprem(r) | \bigcup_{\beta<\alpha} \longrightarrow_{True(Red^\beta(P))} \not\models \psi'\}$. By the induction on α and lemma 5.5, $\forall \beta < \alpha$: $\longrightarrow_{True(Red^\beta(P))} \subseteq \longrightarrow_{Strip(Red^\beta(P),\longrightarrow)} \subseteq \longrightarrow_{Strip(P,\longrightarrow)}$. Hence, if $\bigcup_{\beta<\alpha} \longrightarrow_{True(Red^\beta(P))} \models \psi'$, then $\longrightarrow_{Strip(P,\longrightarrow)} \models \psi'$. Thus $\longrightarrow_{Strip(P,\longrightarrow)} \models prem(r'') = prem(r') \cup \{\psi' \in pprem(r) | \bigcup_{\beta<\alpha} \longrightarrow_{True(Red^\beta(P))} \models \psi'\}$. So $\longrightarrow_{Strip(P,\longrightarrow)} \models conc(r'') = \psi$.

□

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 8.12 shows that the converse is not true: a TSS having a unique stable transition relation need not reduce to a positive TSS.)

Theorem 5.7 *(Soundness of reduction). Let $P = (\Sigma, A, R)$ be a TSS and let $\longrightarrow \subseteq Tr(\Sigma, A)$. For all ordinals α we have:*

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

Proof.

$\Rightarrow$) Let $\longrightarrow = \longrightarrow_{Strip(P,\longrightarrow)}$.

We prove by induction that for all ordinals α:

$$\longrightarrow = \longrightarrow_{Strip(Red^\alpha(P),\longrightarrow)}, \tag{1}$$

$$\longrightarrow_{True(Red^\alpha(P))} \subseteq \longrightarrow \subseteq \longrightarrow_{Pos(Red^\alpha(P))}. \tag{2}$$

By lemma 5.5, always (1) $\Rightarrow$ (2), so we must prove (1).

Basis. $\longrightarrow = \longrightarrow_{Strip(P,\longrightarrow)} = \longrightarrow_{Strip(Red^0(P),\longrightarrow)}$ is given.

Induction. By induction it follows from (2) that for all $\beta < \alpha$:

$$\longrightarrow_{True(Red^\beta(P))} \subseteq \longrightarrow \subseteq \longrightarrow_{Pos(Red^\beta(P))}, \text{ so}$$

$$\bigcup_{\beta<\alpha} \longrightarrow_{True(Red^\beta(P))} \subseteq \longrightarrow \subseteq \bigcap_{\beta<\alpha} \longrightarrow_{Pos(Red^\beta(P))}, \text{ so by lemma 5.6}$$

$$\longrightarrow = \longrightarrow_{Strip(P,\longrightarrow)} = \longrightarrow_{Strip(Red^\alpha(P),\longrightarrow)}.$$

$\Leftarrow$) Let $\longrightarrow = \longrightarrow_{Strip(Red^\alpha(P),\longrightarrow)}$.

Then by lemma 5.5 for all $\beta < \alpha$: $\longrightarrow_{True(Red^\beta(P))} \subseteq \longrightarrow \subseteq {}_{Pos(Red^\beta(P))}$. So again

$$\bigcup_{\beta<\alpha} \longrightarrow_{True(Red^\beta(P))} \subseteq \longrightarrow \subseteq \bigcap_{\beta<\alpha} \longrightarrow_{Pos(Red^\beta(P))}$$

and by lemma 5.6 $\longrightarrow = \longrightarrow_{Strip(P,\longrightarrow)} = \longrightarrow_{Strip(Red^\alpha(P),\longrightarrow)}$.

□

Corollary 5.8 *(Cf. [12], 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 shall answer these questions affirmatively. We show that for a stratified TSS P, the relation $\longrightarrow_P$ as defined in section 3 is stable for P. Furthermore, we show that 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).

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. Therefore, we show in this section that the methods can be used cooperatively, rather than being alternatives for each other.

Finally, we use this amalgamation to demonstrate that the TSS $\mathrm{BPA}_{\delta\epsilon\tau}$ with priorities has an associated transition relation under some conditions.

Theorem 6.1. *If P is stratified, then $\longrightarrow_P$ is stable for P.*
Proof. Let $P = (\Sigma, A, R)$ and let $S : Tr(\Sigma, A) \to \alpha$ be a stratification of P.

1. We show that $\longrightarrow_{Strip(P,\longrightarrow_P)} \subseteq \longrightarrow_P$. Suppose $Strip(P, \longrightarrow_P) \vdash \psi$. We use induction on the structure of the proof of ψ. As $Strip(P, \longrightarrow_P) \vdash \psi$, there exists a rule $r' \in Strip(R, \longrightarrow_P)$ such that $prem(r') \subseteq \longrightarrow_{Strip(P,\longrightarrow_P)}$ and $\psi = conc(r')$. So $\exists r \in R$: $pprem(r) = prem(r')$, $conc(r) = conc(r')$ and $\longrightarrow_P \models nprem(r)$. By induction $pprem(r) \subseteq \longrightarrow_P$. Hence, $\longrightarrow_P \models prem(r)$. As by theorem 3.16 $\longrightarrow_P$ is a model of P, $\psi = conc(r) \in \longrightarrow_P$.

2. Here it is shown that $\longrightarrow_P \subseteq \longrightarrow_{Strip(P,\longrightarrow_P)}$. Recall that $\longrightarrow_P = \bigcup_{0 \le i < \alpha} \longrightarrow_{P_i}$. We show by induction that for every i, $0 \le i < \alpha$: $\longrightarrow_{P_i} \subseteq \longrightarrow_{Strip(P,\longrightarrow_P)}$. Let $\psi \in \longrightarrow_{P_i}$, hence $P_i \vdash \psi$. With induction on the proof of ψ from P_i we show that $Strip(P, \longrightarrow_P) \vdash \psi$.

   Suppose the last rule used to prove ψ from P_i is r'. This means according definition 3.14 that there is a rule $r \in R$ and a substitution $\sigma : V \to T(\Sigma)$ such that

   $$\bigcup_{0 \le j < i} \longrightarrow_{P_j} \models nprem(\sigma(r)) \cup \{\varphi \in pprem(\sigma(r)) | S(\varphi) < i\},$$

   $$r' = \frac{\{\varphi \in pprem(\sigma(r)) | S(\varphi) = i\}}{conc(\sigma(r))}$$

   and $conc(r) = \psi$. As P is stratified, for all $t \xrightarrow{a}\!\!\!\!/ \in nprem(\sigma(r))$ and $t' \in T(\Sigma)$: $S(t \xrightarrow{a} t') < S(\psi) = i$. Thus $\bigcup_{0 \le j < i} \longrightarrow_{P_j} \models nprem(\sigma(r))$ implies $\longrightarrow_P \models nprem(\sigma(r))$ and therefore there is a rule

   $$r'' = \frac{pprem(\sigma(r))}{conc(\sigma(r))} \in Strip(P, \longrightarrow_P).$$

For all $\chi \in prem(r'')$ with $S(\chi) < i$: $\longrightarrow_{P_{S(\chi)}} \models \chi$, so by induction $\longrightarrow_{Strip(P,\longrightarrow_P)} \models \chi$. For all $\chi \in prem(r'')$ with $S(\chi) = i$, it follows with induction on the proof tree that $\longrightarrow_{Strip(P,\longrightarrow_P)} \models \chi$. Hence, $Strip(P, \longrightarrow_P) \vdash prem(r'')$ and hence, $Strip(P, \longrightarrow_P) \vdash conc(r'') = \psi$.

□

Theorem 6.2. *Let $P = (\Sigma, A, R)$ be a TSS with stratification $S : Tr(\Sigma, A) \rightarrow \alpha$. Then $Red^{\alpha}(P)$ is a positive TSS.*
Proof. We show that $\bigcup_{\beta<\alpha} \longrightarrow_{True(Red^{\beta}(P))} = \bigcap_{\beta<\alpha} \longrightarrow_{Pos(Red^{\beta}(P))}$. According to remark 5.2 this is sufficient.

$\subseteq$. This implication follows immediately from lemma 5.5.

$\supseteq$. We claim that for any $\psi \in Tr(\Sigma, A)$:

$$\psi \in \longrightarrow_{Pos(Red^{S(\psi)}(P))} \;\Rightarrow\; \psi \in \longrightarrow_{True(Red^{S(\psi)}(P))}.$$

Using the claim, we can easily finish the proof: as $S(\psi) < \alpha$, we have

$$\begin{aligned}
&\psi \in \textstyle\bigcap_{\beta<\alpha} \longrightarrow_{Pos(Red^{\beta}(P))} \Rightarrow\\
&\psi \in \longrightarrow_{Pos(Red^{S(\psi)}(P))} \Rightarrow\\
&\psi \in \longrightarrow_{True(Red^{S(\psi)}(P))} \Rightarrow\\
&\psi \in \textstyle\bigcup_{\beta<\alpha} \longrightarrow_{True(Red^{S(\psi)}(P))}.
\end{aligned}$$

We prove our claim by transfinite induction on $S(\psi)$. Assume the induction hypothesis holds for all $\gamma < \beta$. Take some $\psi \in Tr(\Sigma, A)$ with $S(\psi) = \beta$. Furthermore, assume $\psi \in \longrightarrow_{Pos(Red^{\beta}(P))}$. Hence, there is a proof of ψ from $Pos(Red^{\beta}(P))$. With induction on this proof, we show that $True(Red^{\beta}(P)) \vdash \psi$. As $Pos(Red^{\beta}(P)) \vdash \psi$, there is a rule $r \in Pos(Red^{\beta}(R))$ such that $conc(r) = \psi$ and $Pos(Red^{\beta}(P)) \vdash prem(r)$. Hence, there is some rule $r' \in Red^{\beta}(R)$ such that $conc(r') = conc(r) = \psi$ and $pprem(r') = prem(r)$. We show that $nprem(r') = \emptyset$. In order to obtain a contradiction, assume $t \overset{a}{\not\longrightarrow} \in nprem(r')$. As $r' \in Red^{\beta}(P)$, we know:

$$\bigcup_{\zeta<\beta} \longrightarrow_{True(Red^{\zeta}(P))} \models t \overset{a}{\not\longrightarrow}.$$

So for every $t' \in T(\Sigma)$: $t \xrightarrow{a} t' \notin \bigcup_{\zeta<\beta} \longrightarrow_{True(Red^{\zeta}(P))}$. In particular, as $S(t \xrightarrow{a} t') = \gamma' < S(\psi) = \beta$, $t \xrightarrow{a} t' \notin \longrightarrow_{True(Red^{\gamma'}(P))}$. By induction, $t \xrightarrow{a} t' \notin \longrightarrow_{Pos(Red^{\gamma'}(P))}$ and so $t \xrightarrow{a} t' \notin \bigcap_{\zeta<\beta} \longrightarrow_{Pos(Red^{\zeta}(P))}$. Therefore,

$$\bigcap_{\zeta<\beta} \longrightarrow_{Pos(Red^{\zeta}(P))} \models t \overset{a}{\not\longrightarrow}.$$

Hence, $t \overset{a}{\not\longrightarrow} \notin nprem(r')$. As $nprem(r') = \emptyset$, $r = r' \in True(Red^{\beta}(R))$. By induction (on the depth of the proof tree of $Pos(Red^{\beta}(P)) \vdash \psi$) we know that $True(Red^{\beta}(P)) \vdash prem(r)$ and thus $True(Red^{\beta}(P)) \vdash \psi$. So we can conclude $\psi \in \longrightarrow_{True(Red^{\beta}(P))}$.

□

Corollary 6.3 *(Cf. [13], corollary 1 and [12], theorem 6.3). Let $P = (\Sigma, A, R)$ be a TSS with stratification $S : Tr(\Sigma, A) \rightarrow \alpha$. Then $\longrightarrow_P = \longrightarrow_{Red^{\alpha}(P)}$ is associated with P.*
Proof. Directly using theorem 6.1, theorem 6.2 and corollary 5.8. □

Lemma 6.4. *Let P be a TSS.*

$$Red^{\alpha}(Red^{\beta}(P)) = Red^{\alpha+\beta}(P).$$

Proof. Straightforward with induction on α, using lemma 5.5. □

Corollary 6.5 *(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.*

Proof. By theorem 6.2 and lemma 6.4 it follows that $Red^{\alpha}(Red^{\beta}(P)) = Red^{\alpha+\beta}(P)$ is a positive TSS. Using corollary 6.3 and lemma 6.4 we have that

$$\longrightarrow_{Red^{\beta}(P)} = \longrightarrow_{Red^{\alpha}(Red^{\beta}(P))} = \longrightarrow_{Red^{\alpha+\beta}(P)}$$

is the transition relation associated with $Red^{\beta}(P)$. Now by theorem 5.7 $\longrightarrow_{Red^{\beta}(P)}$ 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 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 4.8. This is conform the standard practise in process algebra.

2. There is no $a \in Act$ such that $\tau < a$. The motivation for this second condition is threefold (cf. [35] where it is argued that $\tau > a$ for all actions a seems the most 'intuitive' choice).

   - It is essential that τ-actions are not observable. Thus between two observable actions, any number of τ-actions can take place, and must be possible in any process specification. Indeed, the τ-rules R9.1-R9.3 ensure that (in BPA$_{\delta\epsilon\tau}$) every specification satisfies this property. However, allowing $\tau < a$ would destroy this property, as in this case e.g. $\theta(a \cdot a)$ specifies a process performing two a-actions, with no τ-actions in between (assuming there is no $b > a$):

   $$True(P_{\theta}) \vdash \theta(a \cdot a) \xrightarrow{a} \theta(\epsilon \cdot a) \xrightarrow{a} \theta(\epsilon),$$

   but for no $t, t' \in T(\Sigma_{\theta})$:

   $$Pos(Red^{1}(P_{\theta})) \vdash \theta(a \cdot a) \xrightarrow{a} t \xrightarrow{\tau} t'.$$

   (If $\theta(a \cdot a) \xrightarrow{a} t \in \longrightarrow_{Pos(Red^{1}(P_{\theta}))}$, then $t \equiv \theta(u)$ for some u such that $True(P_{\theta}) \vdash u \xrightarrow{a} \epsilon$; every rule in P_{θ} with a conclusion of the form $\theta(u) \xrightarrow{\tau} t'$ has a premise $\theta(u) \xrightarrow{\tau} u'$ (R9.2 and R9.3) or $u \overset{a}{\not\longrightarrow}$ (R5.1)).

   - As a consequence, the axiom $\theta(a \cdot x) = a \cdot \theta(x)$, which is part of the complete axiomatization of BPA$_{\epsilon\delta}$ with priorities (without τ, Cf. [2] axiom TH1 and TH2), is no longer valid: when $\tau < a$, $\theta(a \cdot a)$ cannot perform τ after a although $a \cdot \theta(a)$ can.

   - We conjecture that there is only one transition relation stable for P_{θ}, even for instances with $\tau < a$. However, we have no proof for this. In particular, we do not know whether such an instance of BPA$_{\delta\epsilon\tau}$ with priorities always reduces to a stratified TSS. The problem is caused by the fact that we do not reduce one TSS (with $(Act, <)$ and (Ξ, E) fixed), but try to reduce the whole class of instances of BPA$_{\delta\epsilon\tau}$ with priorities (satisfying condition 1) at once.

Theorem 6.6. *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:

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

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:*

$$t \xrightarrow{a} u \in \longrightarrow_{Pos(P_\theta)} \quad \Rightarrow \quad N(t) \geq N(u)$$

Proof of fact 1. It can be shown for every ground instance r of a rule in P_θ that if for every literal $t \xrightarrow{a} u \in pprem(r)$ $N(t) \geq N(u)$ holds, then $N(t') \geq N(u')$ holds, where $conc(r) = t' \xrightarrow{b} u'$. Instead of giving a detailed treatment of each rule, we only prove the most important ones here:

R4.1 $N(x \cdot y) = \max(N(x), N(y)) \geq N(x) \geq N(x')$ and $\max(N(x), N(y)) \geq N(y)$. This implies that $\max(N(x), N(y)) \geq \max(N(x'), N(y)) = N(x' \cdot y)$.

R8 $N(X) = 0$. So we must prove $N(y) = 0$. Indeed $N(y) \leq N(t_X) = 0$ as by assumption t_X does not contain τ_I-operators. □

For example, the literal $t \xrightarrow{b} \tau_{\{a\}}(\theta(t))$ used in example 3.17 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:

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

(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 $Reduce(P_\theta, \longrightarrow_{True(P_\theta)}, \longrightarrow_{Pos(P_\theta)})$.*

Proof of fact 2. Let r be a ground instance of a rule in $Reduce(P_\theta, \longrightarrow_{True(P_\theta)}, \longrightarrow_{Pos(P_\theta)})$. We must show that for every $\psi \in pprem(r)$: $S(\psi) \leq S(conc(r))$. Furthermore, it must hold that for every $\psi = t \xrightarrow{a}\!\!\!\!\!/\ \in nprem(r)$ and for every $t' \in T(\Sigma)$: $S(t \xrightarrow{a} t') < S(conc(r))$. For most rules this is trivial, as the unreduced instances of the rule already satisfy the requirement. We only consider the most interesting cases:

R5.1 $N(\theta(x)) = N(x)$ implies that $S(\theta(x) \xrightarrow{a} \theta(x')) = S(x \xrightarrow{a} x')$. For each $b > a$ it holds that $S(x \xrightarrow{b} t') < S(x \xrightarrow{a} x')$ for any $t' \in T(\Sigma_\theta)$.

R6.1 $N(x) \leq N(x \lhd y)$, so $S(x \xrightarrow{a} x') \leq S(x \lhd y \xrightarrow{a} x')$. Also $N(y) \leq N(x \lhd y)$, so for each $b > a$ and $t' \in T(\Sigma_\theta)$: $S(y \xrightarrow{b} t') < S(x \lhd y \xrightarrow{a} x')$.

R7.2 $N(\tau_I(x)) = N(x) + 1 > N(x)$. So $S(x\xrightarrow{a}x') < S(\tau_I(x)\xrightarrow{\tau}\tau_I(x'))$.

R9.2 and R9.3 By the first fact $Reduce(P_\theta, \longrightarrow_{True(P_\theta)}, \longrightarrow_{Pos(P_\theta)})$ contains only those instances of these rules for which $N(x) \geq N(y) \geq N(z)$. We need $N(x) \geq N(y)$ to prove e.g. $(y\xrightarrow{\tau}z) \leq (x\xrightarrow{a}z)$, hence $S(y\xrightarrow{\tau}z) \leq S(x\xrightarrow{a}z)$. □

Using corollary 6.5 $\longrightarrow_{Reduce(P_\theta, \longrightarrow_{True(P_\theta)}, \longrightarrow_{Pos(P_\theta)})} = \longrightarrow_{Red^1(P_\theta)}$ is associated with P_θ. □

7 Bisimulation relations

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 ([15]), of which strong bisimulation equivalence is most often used [24, 26]. In this and the subsequent sections 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 $(P\text{-})$*bisimilar*, notation $t \leftrightarroweq_P u$, if there is a strong bisimulation relation R based on P such that tRu. Note that $\leftrightarroweq_P$, the strong bisimulation equivalence *induced* by P, is an equivalence relation.

Thus $t \leftrightarroweq_P u$ means that if t can do some step, u can do a 'similar' step (and vice versa, hence the name *bi*simulation). In the next section we prove that under specific conditions on P, $\leftrightarroweq_P$ is a congruence relation. To this end we shall approximate $\longrightarrow_P$ by other transition relations $\longrightarrow_Q$, and use the notion of $P{\Rightarrow}Q$-bisimulation, meaning that if t can do some step in $\longrightarrow_P$, u can do a 'similar' step in $\longrightarrow_Q$ (and vice versa, i.e. if u can do a step in $\longrightarrow_P$, t can do a 'similar' step in $\longrightarrow_Q$). In the end, the approximation $\longrightarrow_Q$ will be equal to $\longrightarrow_P$. It may be readily checked that in this case, $P{\Rightarrow}Q$-bisimulation is exactly P-bisimulation. Thus showing that for every approximation $\longrightarrow_Q$ $P{\Rightarrow}Q$-bisimulation is a congruence is sufficient to show that P-bisimulation is a congruence.

Formally, we have the following definition.

Definition 7.2. Let $P = (\Sigma, A, R_P)$ and $Q = (\Sigma, A, R_Q)$ be TSS's with associated transition relations $\longrightarrow_P$ and $\longrightarrow_Q$. A relation R is a *strong $P{\Rightarrow}Q$-bisimulation relation* if it satisfies:

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

We say that two terms $t, u \in T(\Sigma)$ are *$P{\Rightarrow}Q$-bisimilar*, notation $t \leftrightarroweq_{P\Rightarrow Q} u$, if there is a strong $P{\Rightarrow}Q$-bisimulation relation R such that tRu. Note that like $\leftrightarroweq_P$, $\leftrightarroweq_{P\Rightarrow Q}$ is reflexive and symmetric. In contrast with $\leftrightarroweq_P$, $\leftrightarroweq_{P\Rightarrow Q}$ need not be transitive.

8 The *ntyft/ntyxt*-format and the congruence theorem

A desirable property for TSS's is that the induced strong bisimulation equivalence is a congruence. In [17] this led to the observation that if a (positive) TSS is in the so-called *tyft/tyxt*-format then this is the case. In [16] 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. In the end of this section we show that 'positive after reduction' is a necessary requirement for the congruence theorem: we give a TSS in *ntyft/ntyxt*-format with a unique stable transition relation for which strong bisimulation is not a congruence.

Definition 8.1. 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:

$$\frac{\{t_k \xrightarrow{a_k} y_k \mid k \in K\} \ \cup \ \{t_l \overset{b_l}{\not\rightarrow} \mid l \in L\}}{f(x_1, ..., x_{rank(f)}) \xrightarrow{a} t}$$

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:

$$\frac{\{t_k \xrightarrow{a_k} y_k \mid k \in K\} \ \cup \ \{t_l \overset{b_l}{\not\rightarrow} \mid l \in L\}}{x \xrightarrow{a} t}$$

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-format* if all its rules are in *ntyft*-format and P is in *ntyft/ntyxt*-format if all its rules are either in *ntyft*- or in *ntyxt*-format.

It may be useful to point out why this format is called the *ntyft/ntyxt*-format. As stated above, the 'n' was added to indicate the possibility of negative premises. The letters *tyft* can be found if one reads first the (positive) premises and then the conclusion from left to right: t represents a term in the left hand side of a premise, y the variable in the right hand side; f is the function name in the left hand side of the conclusion and t the term in the right hand side. Similarly, the other format is called *ntyxt*.

As in [16, 17], we need the following well-foundedness condition in order to prove the congruence theorem.

Definition 8.2 *(Well-foundedness).* Let $P = (\Sigma, A, R)$ be a TSS. Let $S = \{t_k \xrightarrow{a_k} t'_k \mid k \in K\} \subseteq \mathbb{T}(\Sigma) \times A \times \mathbb{T}(\Sigma)$ 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: $\{< x, y > \mid x \in Var(t_k),\ y \in Var(t'_k) \text{ for some } k \in K\}$.

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.

Definition 8.3. Let $P = (\Sigma, A, R)$ be a TSS. Let $r \in R$ be a rule. A variable x is called *free* in r if it occurs in r but not in the source of the conclusion or in the target of a positive premise. The rule r is called *pure* if it is well-founded and does not contain free variables. P is called *pure* if all rules in R are pure.

In what follows we state a number of technicalities needed for the proof of theorem 8.11. At first reading it is advised to skip the remainder of this section except for this theorem.

Definition 8.4. Let S be a set of positive literals which is well-founded and let VDG be the variable dependency graph of S. Let $Var(S)$ be the set of variables occurring in literals in S. Define for each $x \in Var(S)$: $n_{VDG}(x) = sup(\{n_{VDG}(y) + 1 | < y, x > \text{ is an edge of } VDG\})$ $(sup(\emptyset) = 0)$.

Remark 8.5. If S is a set of positive premises of a rule in *ntyft/ntyxt*-format then $n_{VDG}(x) \in \mathbb{N}$ for each $x \in Var(S)$: Every variable y_k occurs only once in the right hand side of a positive literal in the premises. As the term t_k is finite, it contains only a finite number of variables x. Therefore the set $U = \{n_{VDG}(x) + 1 | < x, y_k > \text{ is an edge of } VDG\}$ is finite. Hence, $n_{VDG}(y_k) = sup(U)$ is a natural number.

The following lemma states that any TSS in *ntyft/ntyxt*-format is 'equivalent' to a pure TSS in *ntyft*-format. This allows us to only study *ntyft*-rules.

Lemma 8.6. *Let P be a well-founded TSS in* ntyft/ntyxt-*format and let $\longrightarrow$ be the transition relation associated with P. Then there is a pure TSS P' in* ntyft-*format such that $\longrightarrow$ is also associated with P'. Moreover, P' is positive after reduction iff P is positive after reduction.*
Proof. Assume $P = (\Sigma, A, R)$ and $\Sigma = (F, rank)$. First we construct a TSS $P'' = (\Sigma, A, R'')$ which is pure and in *ntyft/ntyxt*-format. R'' contains a rule $\sigma(r)$ iff r is a rule in R and $\sigma : V \longrightarrow \mathbb{T}(\Sigma)$ is a substitution such that for each variable that is free in r: $\sigma(x) \in T(\Sigma)$ and for each variable x that is not free in r $\sigma(x) = x$. From P'' we construct P' as follows: $P' = (\Sigma, A, R')$ where for each $f \in F$, a rule $\sigma_f(r) \in R'$ iff r is a rule in R'' and $\sigma_f : V \longrightarrow \mathbb{T}(\Sigma)$ is a substitution satisfying:

if r is in *ntyft*-format, then $\sigma_f(z) = z$ for all $z \in V$,
if r is in *ntyxt*-format, then $\sigma_f(z) = z$ for all $z \in V \setminus \{x\}$ and $\sigma_f(x) = f(z_1, ..., z_{rank(f)})$.

Here z_i $(1 \leq i \leq rank(f))$ are variables that do not occur in r. It is easy to see that P' is a pure TSS in *ntyft*-format. Observe that the ground instances of the rules in R, R' and R'' are the same. Also note that 'stable for' and 'positive after reduction' are defined w.r.t. these ground instances. Therefore, $\longrightarrow$ is also the unique transition relation stable for P' and P''. Furthermore P' and P'' are positive after reduction iff P is positive after reduction. □

The relation R_P that is defined now forms the backbone of all remaining proofs in this section.

Definition 8.7. Let $\Sigma = (F, rank)$ be a signature and let $P = (\Sigma, A, R)$ be a TSS with an associated transition relation. The relation $R_P \subseteq T(\Sigma) \times T(\Sigma)$ is the minimal relation satisfying:

- $\leftrightarrows_P \subseteq R_P$,
- for all function names $f \in F$:

$$\forall 1 \leq k \leq rank(f): \; u_k R_P v_k \;\; \Rightarrow \;\; f(u_1, ..., u_{rank(f)}) R_P f(v_1, ..., v_{rank(f)}).$$

Note that this definition is in fact saying that R_P is the minimal congruence relation that includes $\leftrightarrows_P$. This explains the following lemma, which is a standard fact about congruence relations.

Lemma 8.8. *Let $P = (\Sigma, A, R)$ be a TSS with an associated transition relation. Let $t \in \mathbb{T}(\Sigma)$ and let $\sigma, \sigma' : V \longrightarrow T(\Sigma)$ be substitutions such that for all x in $Var(t)$ $\sigma(x) R_P \sigma'(x)$. Then $\sigma(t) R_P \sigma'(t)$.*
Proof. Straightforward with induction on the structure of t. □

Lemma 8.9. *Let P be a pure TSS in ntyft-format. Suppose that $\longrightarrow_P$ is the transition relation that is associated with P. Then for all ordinals $\alpha \geq 0$: R_P is a*

1. *$P \Rightarrow Pos(Red^\alpha(P))$-bisimulation relation.*
2. *$True(Red^\alpha(P)) \Rightarrow P$-bisimulation relation.*

Proof. Assume $P = (\Sigma, A, R)$ and $\Sigma = (F, rank)$. We show the two statements in the lemma by mutual transfinite induction on α.

1. For reasons of symmetry it is enough to show that:

$$\begin{aligned}&\text{if } uR_Pv \text{ and } \longrightarrow_P \models u \xrightarrow{a} u',\\ &\text{then } \exists v' \in T(\Sigma) \text{ such that } \longrightarrow_{Pos(Red^\alpha(P))} \models v \xrightarrow{a} v' \text{ and } u'R_Pv'.\end{aligned}$$

We prove this by induction on the proof of $u \xrightarrow{a} u'$ from $Strip(P, \longrightarrow_P)$. As uR_Pv, two cases arise:

- $u \underline{\leftrightarrow}_P v$. Then $\longrightarrow_P \models u \xrightarrow{a} u'$ implies $\exists v' \in T(\Sigma)$: $\longrightarrow_P \models v \xrightarrow{a} v'$ and $u' \underline{\leftrightarrow}_P v'$. By lemma 5.5 and theorem 5.7 $\longrightarrow_P \subseteq \longrightarrow_{Pos(Red^\alpha(P))}$. So $\longrightarrow_{Pos(Red^\alpha(P))} \models v \xrightarrow{a} v'$. Furthermore, $u' \underline{\leftrightarrow}_P v'$ implies $u'R_Pv'$.
- For some function name $f \in F$, $u = f(u_1, ..., u_{rank(f)})$, $v = f(v_1, ..., v_{rank(f)})$ and $u_iR_Pv_i$ for $1 \leq i \leq rank(f)$. Then there is a rule:

$$r = \frac{\{t_k \xrightarrow{a_k} y_k | k \in K\} \cup \{t_l \stackrel{a_l}{\nrightarrow} | l \in L\}}{f(x_1, ..., x_{rank(f)}) \xrightarrow{a} t} \in R$$

and a substitution σ such that $\sigma(x_i) = u_i$ $(1 \leq i \leq rank(f))$, $\sigma(t) = u'$, $\longrightarrow_P \models prem(\sigma(r))$ and

$$\frac{pprem(\sigma(r))}{conc(\sigma(r))}$$

is the last rule of the proof of $u \xrightarrow{a} u'$ from $Strip(P, \longrightarrow_P)$. Thus the proof of $\sigma(t_k \xrightarrow{a_k} y_k)$ $(k \in K)$ from $Strip(P, \longrightarrow_P)$ is less deep. As P is pure, $\{x_1, .., x_{rank(f)}\} \cup \{y_k | k \in K\} = Var(r)$.

Claim 1. *There is a closed substitution σ' such that for all $x \in Var(r)$:*

(a) $\sigma(x)R_P\sigma'(x)$,

(b) if $x = x_i$ then $\sigma'(x) = v_i$,

(c) if $x = y_k$ $(k \in K)$ then $\sigma'(t_k \xrightarrow{a_k} y_k) \in \longrightarrow_{Pos(Red^\alpha(P))}$,

(d) for all $l \in L$ and for all $\beta < \alpha$: $\longrightarrow_{True(Red^\beta(P))} \models \sigma'(t_l) \stackrel{a_l}{\nrightarrow}$.

Proof of claim 1. We prove the first three points of the claim by inductively constructing $\sigma'(x)$ for every $x \in Var(r)$, using induction on the degree of x in the VDG of $pprem(r)$.
For $x \in \{x_1, ..., x_{rank(f)}\}$, $\sigma'(x_i) = v_i$ is prescribed. Also $\sigma(x_i) = u_iR_Pv_i = \sigma'(x_i)$ is satisfied.
For $x = y_k$ $(k \in K)$, we have $t_k \xrightarrow{a_k} y_k \in pprem(r)$. For all $y \in Var(t_k)$, $n_{VDG}(y) < n_{VDG}(x)$, so by induction $\sigma(y)R_P\sigma'(y)$. As R_P is a congruence, $\sigma(t_k)R_P\sigma'(t_k)$. Since the proof of $\sigma(t_k \xrightarrow{a_k} y_k)$ is less deep than the proof of $u \longrightarrow_a u'$ from $Strip(P, \longrightarrow_P)$, by induction $\exists w \in T(\Sigma)$:$\sigma'(t_k) \xrightarrow{a_k} w \in \longrightarrow_{Pos(Red^\alpha(P))}$ and $\sigma(y_k)R_Pw$. Thus we take $\sigma'(y_k) = w$. Note that the first three points of claim 1 are satisfied which finishes the first part of the proof.
It remains to be shown that $\forall \beta < \alpha$: $\longrightarrow_{True(Red^\beta(P))} \models \sigma'(t_l) \stackrel{a_l}{\nrightarrow}$ $(l \in L)$. Again $\sigma(t_l)R_P\sigma'(t_l)$. Assume to generate a contradiction that $\exists \beta < \alpha$: $\longrightarrow_{True(Red^\beta(P))} \models \sigma'(t_l) \xrightarrow{a_l} s$ for some s. By simultaneous induction $\forall \beta < \alpha$: $\sigma(t_l) \underline{\leftrightarrow}_{True(Red^\beta(P)) \Rightarrow P} \sigma'(t_l)$. So $\longrightarrow_P \models \sigma(t_l) \xrightarrow{a_l} s'$ for some s'. This contradicts the fact that $\longrightarrow_P \models prem(\sigma(r))$. Hence, for all $l \in L$ and for all $\beta < \alpha$: $\longrightarrow_{True(Red^\beta(P))} \models \sigma'(t_l) \stackrel{a_l}{\nrightarrow}$. □

According to claim 1 there is a substitution σ' with the properties (a),(b),(c) and (d). Consider

$$\begin{aligned} V &= Pos(Red^{\alpha}(\{\sigma'(r)\}) \\ &= Pos(Reduce(\{\sigma'(r)\}, \bigcup_{\beta<\alpha} \longrightarrow_{True(Red^{\beta}(P))}, \bigcap_{\beta<\alpha} \longrightarrow_{Pos(Red^{\beta}(P))})). \end{aligned}$$

First we show that $\exists r' \in V$. It follows immediately from clause (d) in the claim that

$$\bigcup_{\beta<\alpha} \longrightarrow_{True(Red^{\beta}(P))} \models nprem(\sigma'(r)).$$

Furthermore, by clause (c) and lemma 5.5:

$$\bigcap_{\beta<\alpha} \longrightarrow_{Pos(Red^{\beta}(P))} \models pprem(\sigma'(r)).$$

Hence, there is some $r' \in V$. It follows from clause (c) in claim 1 that $\longrightarrow_{Pos(Red^{\alpha}(P))} \models pprem(\sigma'(r))$ and therefore, we have that $\longrightarrow_{Pos(Red^{\alpha}(P))} \models pprem(r') = prem(r')$. Thus $\longrightarrow_{Pos(Red^{\alpha}(P))} \models conc(r') = conc(\sigma'(r)) = v \xrightarrow{a} \sigma'(t)$ and $u' = \sigma(t) R_P \sigma'(t) = v'$.

2. For reasons of symmetry it is enough to show that:

   If uR_Pv and $\longrightarrow_{True(Red^{\alpha}(P))} \models u \xrightarrow{a} u'$,
   then $\exists v' \in T(\Sigma)$ such that $\longrightarrow_P \models v \xrightarrow{a} v'$ and $u'R_Pv'$.

As $\longrightarrow_{True(Red^{\alpha}(P))} \models u \xrightarrow{a} u'$, there is a proof tree of $u \xrightarrow{a} u'$ from $True(Red^{\alpha}(P))$. We use induction on the depth of this proof. As uR_Pv we can distinguish two cases:

- $u \leftrightarrows_P v$. As by lemma 5.5 and theorem 5.7 $\longrightarrow_{True(Red^{\alpha}(P))} \subseteq \longrightarrow_P$, $P \models u \xrightarrow{a} u'$. So, $\exists v' \in T(\Sigma)$ such that $\longrightarrow_P \models v \xrightarrow{a} v'$ and $u' \leftrightarrows_P v'$. Hence, $u'R_Pv'$.
- For some function name $f \in F$, $u = f(u_1, ..., u_{rank(f)})$, $v = f(v_1, ..., v_{rank(f)})$ and $u_iR_Pv_i$ for $1 \leq i \leq rank(f)$. In this case the final (ground) rule $r \in True(Red^{\alpha}(R))$ of the proof of $u \xrightarrow{a} u'$ from $True(Red^{\alpha}(P))$ is also present in $Red^{\alpha}(R)$ and has no negative premises.

$$Red^{\alpha}(R) = \begin{cases} Reduce(R_{ground}, \bigcup_{\beta<\alpha} \longrightarrow_{True(Red^{\beta}(P))}, \bigcap_{\beta<\alpha} \longrightarrow_{Pos(Red^{\beta}(P))}) & \text{if } \alpha > 0, \\ R_{ground} & \text{if } \alpha = 0. \end{cases}$$

Thus there is a rule $r' \in R$ and a substitution $\sigma : V \to T(\Sigma)$ such that $\sigma(r')$ is reduced to r. This means that $conc(r) = conc(\sigma(r'))$ and $prem(r) \subseteq pprem(\sigma(r'))$. Moreover, all negative premises of $\sigma(r')$ and all premises in $pprem(\sigma(r')) - pprem(r)$, which are removed, are redundant:

$$\bigcup_{\beta<\alpha} \longrightarrow_{True(Red^{\beta}(P))} \models pprem(\sigma(r')) - pprem(r),$$

$$\bigcap_{\beta<\alpha} \longrightarrow_{Pos(Red^{\beta}(P))} \models nprem(\sigma(r')).$$

As P is in *ntyft*-format, r' is of the form

$$\frac{\{t_k \xrightarrow{a_k} y_k | k \in K\} \cup \{t_l \overset{a_l}{\not\longrightarrow} | l \in L\}}{f(x_1, ..., x_{rank(f)}) \xrightarrow{a} t}$$

and $\sigma(x_i) = u_i$ $(1 \leq i \leq rank(f))$, hence $\sigma(f(x_1, ..., x_{rank(f)})) = u$, and $\sigma(t) = u'$. As P is pure, $\{x_1, ..., x_{rank(f)}\} \cup \{y_k | k \in K\} = Var(r')$.

Claim 2. *There is a closed substitution σ' such that for all $x \in Var(r')$:*

(a) $\sigma(x) R_P \sigma'(x)$,

(b) if $x = x_i$ then $\sigma'(x) = v_i$,

(c) if $x = y_k$ $(k \in K)$ then $\sigma'(t_k) \xrightarrow{a_k} \sigma'(y_k) \in \longrightarrow_P$,

(d) for all $l \in L$: $\longrightarrow_P \models \sigma'(t_l) \overset{a_l}{\not\longrightarrow}$.

Proof of claim 2. We prove the first three points of the claim by giving a construction of $\sigma'(x)$ for every $x \in Var(r')$, using induction on the degree of x in the VDG of $pprem(r')$. For $x \in \{x_1, \ldots, x_{rank(f)}\}$, $\sigma'(x_i) = v_i$ is prescribed. Also $\sigma(x_i) = u_i R_P v_i = \sigma'(x_i)$ is satisfied.
For $x = y_k$ $(k \in K)$, we have $t_k \xrightarrow{a_k} y_k \in pprem(r')$. For all $y \in Var(t_k)$, $n_{VDG}(y) < n_{VDG}(x)$, so by induction $\sigma(y) R_P \sigma'(y)$. As R_P is a congruence, $\sigma(t_k) R_P \sigma'(t_k)$. Two cases arise.

i. $\sigma(t_k \xrightarrow{a_k} y_k) \in pprem(r)$. Then there is a proof of $\sigma(t_k \xrightarrow{a_k} y_k)$ from $True(Red^\alpha(P))$ that is less deep than the proof of $u \xrightarrow{a} u'$. As $\sigma(t_k \xrightarrow{a_k} y_k) \in \longrightarrow_{True(Red^\alpha(P))}$ and $\sigma(t_k) R_P \sigma'(t_k)$, it follows by induction that $\exists w \in T(\Sigma)$: $\sigma'(t_k) \xrightarrow{a_k} w \in \longrightarrow_P$ and $\sigma(y_k) R_P w$.

ii. $\sigma(t_k \xrightarrow{a_k} y_k) \notin pprem(r)$. Hence $\exists \beta < \alpha$: $\longrightarrow_{True(Red^\beta(P))} \models \sigma(t_k \xrightarrow{a_k} y_k)$. As also $\sigma(t_k) R_P \sigma'(t_k)$, it follows by induction that $\exists w \in T(\Sigma)$: $\sigma'(t_k) \xrightarrow{a_k} w \in \longrightarrow_P$ and $\sigma(y_k) R_P w$.

In both cases, we take $\sigma'(y_k) = w$. Note that the first three points of claim 2 are satisfied, which finishes the first part of this proof.
We are left to show that $\longrightarrow_P \models \sigma'(t_l) \overset{a_l}{\not\longrightarrow}$ $(l \in L)$. As $\sigma(t_l) R_P \sigma'(t_l)$, it follows from point 1 of this lemma that $\sigma(t_l) \underline{\leftrightarrow}_{P \Rightarrow Pos(Red^\alpha(P))} \sigma'(t_l)$. In order to obtain a contradiction, assume that $\longrightarrow_P \models \sigma'(t_l) \xrightarrow{a_l} s'$ for some s'. Then $\longrightarrow_{Pos(Red^\alpha(P))} \models \sigma(t_l) \xrightarrow{a_l} s$ for some s. So by lemma 5.5 for all $\beta < \alpha$: $\longrightarrow_{Pos(Red^\beta(P))} \models \sigma(t_l) \xrightarrow{a_l} s$. This cannot be the case, as $\bigcap_{\beta<\alpha} \longrightarrow_{Pos(Red^\beta(P))} \models \sigma(t_l) \overset{a_l}{\not\longrightarrow}$. □

From claim 2 it follows that there is a substitution σ' such that $\longrightarrow_P \models prem(\sigma'(r'))$. Hence $\longrightarrow_P \models conc(\sigma'(r')) = v \xrightarrow{a} \sigma'(t)$. Finally, as for all $x \in Var(r')$: $\sigma(x) R_P \sigma'(x)$, $u' = \sigma(t) R_P \sigma'(t) = v'$.

□

Lemma 8.10. *Let P be a pure TSS in ntyft-format that is positive after reduction. Then $R_P = \underline{\leftrightarrow}_P$.*
Proof. As P is positive after reduction for some ordinal α, $Red^\alpha(P)$ is positive. It follows using corollary 5.8 that:

$$\longrightarrow_P = \longrightarrow_{Red^\alpha(P)} = \longrightarrow_{Pos(Red^\alpha(P))}.$$

Now, it follows using lemma 8.9, the introduction of definition 7.2 and the definition of R_P that:

$$R_P \subseteq \underline{\leftrightarrow}_{P \Rightarrow Pos(Red^\alpha(P))} = \underline{\leftrightarrow}_P \subseteq R_P.$$

□

Theorem 8.11 *(Congruence theorem). Let P be a well-founded TSS in ntyft/ntyxt-format that is positive after reduction. Then $\underline{\leftrightarrow}_P$ is a congruence.*
Proof. Assume $P = (\Sigma, A, R)$. According to lemma 8.6 there is a pure TSS $P' = (\Sigma, A, R')$ in *ntyft*-format that is positive after reduction such that $\longrightarrow_P = \longrightarrow_{P'}$. Hence, $\underline{\leftrightarrow}_P = \underline{\leftrightarrow}_{P'}$. By lemma 8.10 $\underline{\leftrightarrow}_{P'} = R_{P'}$. As $R_{P'}$ is a congruence w.r.t. Σ, $\underline{\leftrightarrow}_P$ is also a congruence w.r.t. Σ. □

The next example shows that the requirement in the congruence theorem 8.11 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 8.12. 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:

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

$$\text{E3:}\quad \frac{x \xrightarrow{a} y \quad f(x) \overset{b_1}{\nrightarrow} \quad f(c_1) \overset{b_2}{\nrightarrow}}{f(x) \xrightarrow{b_2} c_2} \qquad \text{E4:}\quad \frac{x \xrightarrow{a} y \quad f(x) \overset{b_2}{\nrightarrow} \quad f(c_2) \overset{b_1}{\nrightarrow}}{f(x) \xrightarrow{b_1} c_1}.$$

Note that P is pure and in *ntyft*-format. $Red^1(P)$ is a TSS with the following rules:

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

$$\text{E3}':\quad \frac{f(c_1) \overset{b_1}{\nrightarrow} \quad f(c_1) \overset{b_2}{\nrightarrow}}{f(c_1) \xrightarrow{b_2} c_2} \qquad \text{E3}'':\quad \frac{f(c_2) \overset{b_1}{\nrightarrow} \quad f(c_1) \overset{b_2}{\nrightarrow}}{f(c_2) \xrightarrow{b_2} c_2}$$

$$\text{E4}':\quad \frac{f(c_1) \overset{b_2}{\nrightarrow} \quad f(c_2) \overset{b_1}{\nrightarrow}}{f(c_1) \xrightarrow{b_1} c_1} \qquad \text{E4}'':\quad \frac{f(c_2) \overset{b_2}{\nrightarrow} \quad f(c_2) \overset{b_1}{\nrightarrow}}{f(c_2) \xrightarrow{b_1} c_1}.$$

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 $f(c_1) \overset{b_2}{\nrightarrow}$ and $f(c_2) \overset{b_1}{\nrightarrow}$), 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.

9 Conservative extensions of TSS's

It can be useful to enrich a given language with additional language constructs (like in our running example, where $\text{BPA}_{\delta\epsilon\tau}$ is enriched with the priority and unless operator). For these new constructs operational rules are devised which are added to the operational semantics of the basic language. In this section we study how an operational semantics can be extended and especially how we can guarantee that transitions between terms in the basic language are not effected by the extension.

In this section we assume that the operational semantics of the basic language is given by a TSS P_0. All extensions, i.e. the added signature, label set and operational rules are given in a TSS P_1. The extension of P_0 with P_1 is written as $P_0 \oplus P_1$ [17]. Due to the symmetric nature – we could as well extend P_1 with P_0 – this is called the *sum* of P_0 and P_1.

Definition 9.1. Let $\Sigma_i = (F_i, rank_i)$ $(i = 0, 1)$ be two signatures such that for all $f \in F_0 \cap F_1$: $rank_0(f) = rank_1(f)$. The *sum* of Σ_0 and Σ_1, notation $\Sigma_0 \oplus \Sigma_1$, is the signature:

$$\Sigma_0 \oplus \Sigma_1 = (F_0 \cup F_1, \lambda f.\text{if } f \in F_0 \text{ then } rank_0(f) \text{ else } rank_1(f)).$$

Definition 9.2. Let $P_i = (\Sigma_i, A_i, R_i)$ $(i = 0, 1)$ be two TSS's with $\Sigma_0 \oplus \Sigma_1$ defined. The *sum* of P_0 and P_1, notation $P_0 \oplus P_1$, is the TSS:

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

If P_0 is extended with P_1 such that 'the properties' of P_0 are maintained, $P_0 \oplus P_1$ is said to be a *conservative extension* of P_0. With 'properties' of P_0 we mean transitions that can be performed by terms over the signature of P_0. To be more precise:

Definition 9.3. Let $P_i = (\Sigma_i, A_i, R_i)$ $(i = 0, 1)$ be two TSS's such that P_0 has associated transition relation $\longrightarrow_{P_0}$. Let $P_0 \oplus P_1$ with associated transition relation $\longrightarrow_{P_0 \oplus P_1}$ be defined. We say that $P_0 \oplus P_1$ is a *conservative extension* of P_0 and that P_1 *can be added conservatively* to P_0 if

$$\longrightarrow_{P_0 \oplus P_1} \cap \, (T(\Sigma_0) \times (A_0 \cup A_1) \times T(\Sigma_0 \oplus \Sigma_1)) = \longrightarrow_{P_0}.$$

An alternative definition has been given in [17]. Adapting that definition to our terminology, it says that $P = P_0 \oplus P_1 = (\Sigma, A, R)$ with associated transition relation $\longrightarrow_P$ is a conservative extension of $P_0 = (\Sigma_0, A_0, R_0)$ if for all $t \in T(\Sigma_0)$, $a \in A$ and $t' \in T(\Sigma)$:

$$\longrightarrow_P \models t \xrightarrow{a} t' \quad \Leftrightarrow \quad \longrightarrow_{P_0} \models t \xrightarrow{a} t'.$$

where $\longrightarrow_{P_0}$ is associated with P_0.

We now head for a theorem that gives conditions under which P_1 can be added conservatively to P_0. It turns out that this is the case if P_0 is pure and each rule in P_1 contains a function name in the source of its conclusion that does not appear in the signature of P_0. This theorem considerably extends the results in [16] in which a comparable theorem was proved for TSS's in *ntyft/ntyxt*-format. If our result is restricted to this format, both results coincide, except that here, we deal with TSS's that are positive after reduction while in [16] only stratified TSS's were considered.

Lemma 9.4. *Let $\Sigma_0 = (F_0, rank_0)$ be a signature. Let $P_0 = (\Sigma_0, A_0, R_0)$ be a pure TSS and let $P_1 = (\Sigma_1, A_1, R_1)$ be a TSS such that $P_0 \oplus P_1$ is defined and for each rule $r \in R_1$: $source(conc(r)) \notin \mathbb{T}(\Sigma_0)$. Then, for each ordinal α:*

$$\longrightarrow_{Pos(Red^{\alpha}(P_0 \oplus P_1))} \cap \, (T(\Sigma_0) \times (A_0 \cup A_1) \times T(\Sigma_0 \oplus \Sigma_1)) = \longrightarrow_{Pos(Red^{\alpha}(P_0))} \tag{1}$$

$$\longrightarrow_{True(Red^{\alpha}(P_0 \oplus P_1))} \cap \, (T(\Sigma_0) \times (A_0 \cup A_1) \times T(\Sigma_0 \oplus \Sigma_1)) = \longrightarrow_{True(Red^{\alpha}(P_0))} \tag{2}$$

Proof. We prove clauses (1) and (2) by simultaneous induction on α.

(1)$\subseteq$ For this case it is sufficient to show the following:

$$Pos(Red^{\alpha}(P_0 \oplus P_1)) \vdash t \xrightarrow{a} t' \text{ and } t \in T(\Sigma_0) \text{ implies}$$

$$Pos(Red^{\alpha}(P_0)) \vdash t \xrightarrow{a} t', \; a \in A_0 \text{ and } t' \in T(\Sigma_0).$$

So assume that $Pos(Red^{\alpha}(P_0 \oplus P_1)) \vdash t \xrightarrow{a} t'$ and $t \in T(\Sigma_0)$. We use induction on the depth of this proof. Let the last rule of this proof be $r \in Pos(Red^{\alpha}(R_0 \oplus R_1))$. Then $conc(r) = t \xrightarrow{a} t'$. Hence, as $t \in T(\Sigma_0)$ and all rules in R_1 contain a function name $f \notin \Sigma_0$ in the source of their conclusions, r is derived from a rule $\sigma(r')$ with $r' \in R_0$. So $a \in A_0$.

Claim 1. *For all $x \in Var(r')$: $\sigma(x) \in T(\Sigma_0)$.*

Proof of claim 1. As r' is pure, it is well-founded, so $pprem(r')$ has a variable dependency graph VDG. We prove the claim by induction on $n_{VDG}(x)$. Consider some x with $n_{VDG}(x) = \gamma$ and assume the claim holds for all x' such that $n_{VDG}(x') < \gamma$. As r' does not contain free variables, one of the following two cases must hold:

1. $x \in Var(source(conc(r'))$. As $\sigma(source(conc(r')) \in T(\Sigma_0)$, $\sigma(x) \in T(\Sigma_0)$.
2. $x \in Var(u')$ and $u \xrightarrow{a} u' \in pprem(r')$. For all $x' \in Var(u)$: $n_{VDG}(x') < n_{VDG}(x)$ and therefore $\sigma(x') \in T(\Sigma_0)$. Hence, $\sigma(u) \in T(\Sigma_0)$. Distinguish the following two cases:

   (a)
$$\bigcup_{\beta<\alpha} \longrightarrow_{True(Red^\beta(P_0 \oplus P_1))} \models \sigma(u) \xrightarrow{a} \sigma(u').$$

   Then by (2),
$$\bigcup_{\beta<\alpha} \longrightarrow_{True(Red^\beta(P_0))} \models \sigma(u) \xrightarrow{a} \sigma(u')$$
   and this means that $\sigma(u') \in T(\Sigma_0)$. Therefore, as $x \in Var(u')$, $\sigma(x) \in T(\Sigma_0)$.

   (b)
$$\bigcup_{\beta<\alpha} \longrightarrow_{True(Red^\beta(P_0 \oplus P_1))} \not\models \sigma(u) \xrightarrow{a} \sigma(u').$$

   Then $\sigma(u) \xrightarrow{a} \sigma(u') \in pprem(r)$ and therefore,
$$Pos(Red^\alpha(P_0 \oplus P_1)) \vdash \sigma(u) \xrightarrow{a} \sigma(u').$$

   By induction (on the proof tree) it follows that:
$$Pos(Red^\alpha(P_0)) \vdash \sigma(u) \xrightarrow{a} \sigma(u')$$
   and $\sigma(u') \in T(\Sigma_0)$. Hence, $\sigma(x) \in T(\Sigma_0)$. □

As r is derived from reducing $\sigma(r')$ we have the following:

$$\begin{array}{l}\bigcap_{\beta<\alpha} \longrightarrow_{Pos(Red^\beta(P_0 \oplus P_1))} \models pprem(\sigma(r')),\\ \bigcup_{\beta<\alpha} \longrightarrow_{True(Red^\beta(P_0 \oplus P_1))} \models nprem(\sigma(r')),\\ \bigcup_{\beta<\alpha} \longrightarrow_{True(Red^\beta(P_0 \oplus P_1))} \models pprem(\sigma(r')) - prem(r).\end{array}$$

As by claim 1 $\sigma : Var(r') \to T(\Sigma_0)$ it follows using the outermost induction hypothesis that:

$$\begin{array}{l}\bigcap_{\beta<\alpha} \longrightarrow_{Pos(Red^\beta(P_0))} \models pprem(\sigma(r')),\\ \bigcup_{\beta<\alpha} \longrightarrow_{True(Red^\beta(P_0))} \models nprem(\sigma(r')),\\ \bigcup_{\beta<\alpha} \longrightarrow_{True(Red^\beta(P_0))} \models pprem(\sigma(r')) - prem(r).\end{array}$$

Or in other words $r \in Pos(Red^\alpha(R_0))$. By induction on the proof tree and claim 1 it follows that $Pos(Red^\alpha(P_0)) \vdash prem(r)$ and therefore $Pos(Red^\alpha(P_0)) \vdash t \xrightarrow{a} t' = \sigma(conc(r')) \in Tr(\Sigma_0, A_0)$.

(1)$\supseteq$ For this case it is sufficient to prove (using induction on the proof tree for $Pos(Red^\alpha(P_0)) \vdash t \xrightarrow{a} t'$) that:
$$Pos(Red^\alpha(P_0)) \vdash t \xrightarrow{a} t' \quad \Rightarrow \quad Pos(Red^\alpha(P_0 \oplus P_1)) \vdash t \xrightarrow{a} t'.$$

So assume $r \in Pos(Red^\alpha(R_0))$ is the last rule used in the proof for $t \xrightarrow{a} t'$. Hence there is a rule $r' \in R_0$ and a substitution $\sigma : Var(r') \to T(\Sigma_0)$ with $conc(\sigma(r')) = conc(r)$, $prem(r) \subseteq pprem(\sigma(r'))$. Moreover:

$$\begin{array}{l}\bigcap_{\beta<\alpha} \longrightarrow_{Pos(Red^\beta(P_0))} \models pprem(\sigma(r')),\\ \bigcup_{\beta<\alpha} \longrightarrow_{True(Red^\beta(P_0))} \models nprem(\sigma(r')),\\ \bigcup_{\beta<\alpha} \longrightarrow_{True(Red^\beta(P_0))} \models pprem(\sigma(r')) - prem(r).\end{array}$$

As for each premise $\psi \in prem(\sigma(r'))$, $source(\psi) \in T(\Sigma_0)$, we have by induction:

$$\begin{aligned}
&\textstyle\bigcap_{\beta<\alpha} \longrightarrow_{Pos(Red^{\beta}(P_0 \oplus P_1))} \models pprem(\sigma(r')),\\
&\textstyle\bigcup_{\beta<\alpha} \longrightarrow_{True(Red^{\beta}(P_0 \oplus P_1))} \models nprem(\sigma(r')),\\
&\textstyle\bigcup_{\beta<\alpha} \longrightarrow_{True(Red^{\beta}(P_0 \oplus P_1))} \models pprem(\sigma(r')) - prem(r).
\end{aligned}$$

Hence, $r \in Pos(Red^{\alpha}(R_0 \oplus R_1))$. As $Pos(Red^{\alpha}(P_0)) \vdash t \xrightarrow{a} t'$, $Pos(Red^{\alpha}(P_0)) \vdash \psi$ for each $\psi \in prem(r)$. By induction $Pos(Red^{\alpha}(P_0 \oplus P_1)) \vdash \psi$ and hence, $Pos(Red^{\alpha}(P_0 \oplus P_1)) \vdash t \xrightarrow{a} t'$.

(2) This case can be shown in the same way as (1).

□

Theorem 9.5 *(Conservativity). Let $\Sigma_0 = (F_0, rank_0)$ be a signature. Let $P_0 = (\Sigma_0, A_0, R_0)$ be a pure TSS and let $P_1 = (\Sigma_1, A_1, R_1)$ be a TSS such that each rule $r \in R_1$ contains at least one function name $f \notin F_0$ in the source of its conclusion. Furthermore, assume that $P_0 \oplus P_1$ exists and is positive after reduction. Then $P_0 \oplus P_1$ is a conservative extension of P_0.*

Proof. As $P_0 \oplus P_1$ is positive after reduction, there is some ordinal α such that $Red^{\alpha}(P_0 \oplus P_1)$ is a positive TSS. Hence, $P_0 \oplus P_1$ has an associated transition relation $\longrightarrow_{P_0 \oplus P_1}$. Let $A = A_0 \cup A_1$ and $\Sigma = \Sigma_0 \oplus \Sigma_1$. By lemma 9.4 we have:

$$\begin{aligned}
&\longrightarrow_{Pos(Red^{\alpha}(P_0))} =\\
&\longrightarrow_{Pos(Red^{\alpha}(P_0 \oplus P_1))} \cap (T(\Sigma_0) \times A \times T(\Sigma)) =\\
&\longrightarrow_{True(Red^{\alpha}(P_0 \oplus P_1))} \cap (T(\Sigma_0) \times A \times T(\Sigma)) =\\
&\longrightarrow_{True(Red^{\alpha}(P_0))}.
\end{aligned}$$

Hence by remark 5.2, $Red^{\alpha+1}(P_0)$ is a positive TSS. Therefore P_0 also has an associated transition relation $\longrightarrow_{P_0}$. Moreover, using corollary 5.8 and lemma 9.4 we have:

$$\begin{aligned}
&\longrightarrow_{P_0} =\\
&\longrightarrow_{True(Red^{\alpha+1}(P_0))} =\\
&\longrightarrow_{True(Red^{\alpha+1}(P_0 \oplus P_1))} \cap (T(\Sigma_0) \times A \times T(\Sigma)) =\\
&\longrightarrow_{P_0 \oplus P_1} \cap (T(\Sigma_0) \times A \times T(\Sigma)).
\end{aligned}$$

□

Remark 9.6. From the alternative definition of conservativity it is immediately obvious that if $P_0 \oplus P_1$ is a conservative extension of $P_0 = (\Sigma_0, A_0, R_0)$ then for all $t, u \in T(\Sigma_0)$: $t \leftrightarrows_{P_0} u \Leftrightarrow t \leftrightarrows_{P_0 \oplus P_1} u$.

Example 9.7. We can apply the conservativity theorem to show that the priority operator and the unless operator form a conservative extension of $BPA_{\delta\epsilon\tau}$. We can also conservatively add the parallel operator which is characterized by the following rules

$$10.1 \quad \frac{x \xrightarrow{a} x'}{x \parallel y \xrightarrow{a} x' \parallel y} \qquad\qquad 10.2 \quad \frac{y \xrightarrow{a} y'}{x \parallel y \xrightarrow{a} x \parallel y'}$$

to $BPA_{\delta\epsilon\tau}$ with priorities. In fact in almost all cases the addition of new operators to an existing TSS turns out to be conservative.

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

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

10 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 6.6 P_θ has an associated transition relation $\longrightarrow_{P_\theta}$.

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 [2, 24], adding only the axiom P3 showing the interaction between $\triangleleft$ and τ. Nevertheless, as far as we know, this straightforward compilation has not been justified in bisimulation semantics. Only in [35] τ and θ have been combined using an isomorphic embedding.

This section is added to show how an axiom system can be proved sound and complete with respect to an operational semantics, even if this semantics is defined using negative premises. We give all essential lemmas and theorems but only some insightful parts of the proofs. Most proofs apply induction on proof trees (standard for positive TSS's) of the 'stripped' TSS. 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.

Definition 10.1. 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 $tR_{Eq}u$.

The following lemma says how behavior of a complex term can be explained in terms of necessary behavior of its components. This lemma is first used in [34] to prove the soundness of the axioms. Due to the rules R9.2 and R9.3, the proof of this lemma is lengthy.

Lemma 10.2 *(Structuring lemma). Let $t, u, v \in T(\Sigma_\theta)$ and $a \in A_\theta$.*
If $t + u \xrightarrow{a} v$ then one of the following must hold:

1. *$t \xrightarrow{a} v$,*
2. *$u \xrightarrow{a} v$.*

If $t \cdot u \xrightarrow{a} v$ then one of the following must hold:

1. *$t \xrightarrow{a} t'$, $v \equiv t' \cdot u$ and $a \not\equiv \surd$ for some $t' \in T(\Sigma_\theta)$,*
2. *$t \xrightarrow{\surd} t'$ and $u \xrightarrow{a} v$ for some $t' \in T(\Sigma_\theta)$,*
3. *$t \xrightarrow{a} t'$, $t' \xrightarrow{\surd} t''$, $u \xrightarrow{\tau} v$ and $a \not\equiv \surd$ for some $t', t'' \in T(\Sigma_\theta)$.*

If $\theta(t) \xrightarrow{a} u$ then one of the following must hold:

1. *$t \xrightarrow{a} t'$, $u \equiv \theta(t')$, $a \not\equiv \surd$ and $\forall b > a\ t \overset{b}{\not\rightarrow}$ for some $t' \in T(\Sigma_\theta)$,*
2. *$t \xrightarrow{\tau} t'$, $t' \xrightarrow{a} t''$, $u \equiv \theta(t'')$, $a \not\equiv \surd$ and $\forall b > a\ t' \overset{b}{\not\rightarrow}$ for some $t', t'' \in T(\Sigma_\theta)$,*
3. *$t \xrightarrow{\surd} t'$, $u \equiv \theta(t')$ and $a \equiv \surd$ for some $t' \in T(\Sigma_\theta)$.*

If $t \triangleleft u \xrightarrow{a} v$ then one of the following must hold:

1. *$t \xrightarrow{a} v$, $a \not\equiv \surd$ and $\forall b > a\ u \overset{b}{\not\rightarrow}$,*
2. *$t \xrightarrow{\tau} t'$, $t' \xrightarrow{a} v$ and $a \not\equiv \surd$ for some $t' \in T(\Sigma_\theta)$,*
3. *$t \xrightarrow{\surd} v$ and $a \equiv \surd$.*

If $\tau_I(t) \xrightarrow{a} u$ then one of the following must hold:

1. *$t \xrightarrow{a_1} t_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} t_n \xrightarrow{a} t_{n+1} \xrightarrow{a_{n+2}} \ldots \xrightarrow{a_m} t_m$, $a \notin I$ and $u \equiv \tau_I(t_m)$ for some $a_1, .., a_n, a_{n+2}, .., a_m \in I$, $t_1, .., t_m \in T(\Sigma_\theta)$, $n \geq 0$ and $m \geq 1$.*
2. *$t \xrightarrow{a_1} t_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} t_n$, $a \equiv \tau$ and $u \equiv \tau_I(t_n)$ for some $a_1, .., a_n \in I$, $t_1, .., t_n \in T(\Sigma_\theta)$ and $n \geq 1$.*

Proof. As an illustration, we give the proof for $\theta(t) \xrightarrow{a} u$ in case $a \not\equiv \surd$. All other proofs can be given in the same way.

If $\theta(t) \xrightarrow{a} u$ then this is equivalent to saying that $Strip(P_\theta, \longrightarrow_{P_\theta}) \vdash \theta(t) \xrightarrow{a} u$. We show with induction on the proof tree that $Strip(P_\theta, \longrightarrow_{P_\theta}) \vdash \theta(t) \xrightarrow{a} u$ implies that one of the following holds:

1. $t \xrightarrow{a} t'$, $u \equiv \theta(t')$ and $\forall b > a\ t \overset{b}{\not\rightarrow}$ for some $t' \in T(\Sigma_\theta)$,
2. $t \xrightarrow{\tau} t'$, $t' \xrightarrow{a} t''$, $u \equiv \theta(t'')$ and $\forall b > a\ t' \overset{b}{\not\rightarrow}$ for some $t', t'' \in T(\Sigma_\theta)$.

Suppose $Strip(P_\theta, \longrightarrow_{P_\theta}) \vdash \theta(t) \xrightarrow{a} u$. The last rule that is used in this proof must either be R9.2, R9.3 or a stripped version of R5.1. Suppose a simplified version of rule R5.1 has been used. In this case the premises of R5.1, $t \xrightarrow{a} t'$ and $\forall b > a\ t \overset{b}{\not\rightarrow}$, hold in $\longrightarrow_{P_\theta}$. Furthermore, $u \equiv \theta(t')$. So case 1 of θ in the structuring lemma must hold.

If rule R9.2 has been used, we know that $Strip(P_\theta, \longrightarrow_{P_\theta}) \vdash \theta(t) \xrightarrow{\tau} u'$ and $Strip(P_\theta, \longrightarrow_{P_\theta}) \vdash u' \xrightarrow{a} u$. By induction one of the following four cases must hold:

1. $t \xrightarrow{\tau} t'$, $t' \xrightarrow{a} t''$, $\forall b > a\ t' \overset{b}{\not\rightarrow}$ and $u \equiv \theta(t'')$,
2. $t \xrightarrow{\tau} t'$, $t' \xrightarrow{\tau} t''$, $t'' \xrightarrow{a} t'''$, $\forall b > a\ t'' \overset{b}{\not\rightarrow}$ and $u \equiv \theta(t''')$,
3. $t \xrightarrow{\tau} t'$, $t' \xrightarrow{\tau} t''$, $t'' \xrightarrow{a} t'''$, $\forall b > a\ t'' \overset{b}{\not\rightarrow}$ and $u \equiv \theta(t''')$,

4. $t\xrightarrow{\tau}t'$, $t'\xrightarrow{\tau}t''$, $t''\xrightarrow{\tau}t'''$, $t'''\xrightarrow{a}t''''$, $\forall b > a\ t'''\overset{b}{\not\rightarrow}$ and $u \equiv \theta(t'''')$.

In all cases it must hold that for some v and v':

$$t\xrightarrow{\tau}v,\ v\xrightarrow{a}v',\ \forall b > a\ v\overset{b}{\not\rightarrow} \text{ and } u \equiv \theta(v').$$

Suppose rule R9.3 has been used as last step in the proof. As the premises of R9.3 are derivable, we have:

$$Strip(P_\theta, \longrightarrow_{P_\theta}) \vdash \theta(t)\xrightarrow{a}u',\ Strip(P_\theta, \longrightarrow_{P_\theta}) \vdash u'\xrightarrow{\tau}u.$$

By induction one of the following four cases must hold.

1. $t\xrightarrow{a}t'$, $t'\xrightarrow{\tau}t''$, $\forall b > a\ t\overset{b}{\not\rightarrow}$ and $u \equiv \theta(t'')$,
2. $t\xrightarrow{a}t'$, $t'\xrightarrow{\tau}t''$, $t''\xrightarrow{\tau}t'''$, $\forall b > a\ t\overset{b}{\not\rightarrow}$ and $u \equiv \theta(t''')$,
3. $t\xrightarrow{\tau}t'$, $t'\xrightarrow{a}t''$, $t''\xrightarrow{\tau}t'''$, $\forall b > a\ t'\overset{b}{\not\rightarrow}$ and $u \equiv \theta(t''')$,
4. $t\xrightarrow{\tau}t'$, $t'\xrightarrow{a}t''$, $t''\xrightarrow{\tau}t'''$, $t'''\xrightarrow{\tau}t''''$, $\forall b > a\ t'\overset{b}{\not\rightarrow}$ and $u \equiv \theta(t'''')$.

From cases 1 or 2 it follows that (for appropriate $v \in T(\Sigma_\theta)$):

$$t\xrightarrow{a}v,\ \forall b > a\ t\overset{b}{\not\rightarrow} \text{ and } u \equiv \theta(v)$$

which is case 1 for θ in the structuring lemma. From cases 3 or 4 it follows that (for appropriate $v, v' \in T(\Sigma_\theta)$):

$$t\xrightarrow{\tau}v\xrightarrow{a}v',\ \forall b > a\ v\overset{b}{\not\rightarrow} \text{ and } u \equiv \theta(v')$$

which is case 2 for θ in the structuring lemma. □

With the structuring lemma it is rather straightforward, but unpleasantly lengthy, to prove the soundness of the axioms.

Theorem 10.3 *(Soundness of* $\mathrm{BPA}^{\theta}_{\delta\epsilon\tau}$*). Let* $t, u \in T(\Sigma_\theta)$*:*

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

Proof. We must show that $R_{\mathrm{BPA}^{\theta}_{\delta\epsilon\tau}} \subseteq \leftrightarrows_{P_\theta}$. As $R_{\mathrm{BPA}^{\theta}_{\delta\epsilon\tau}}$ is the smallest congruence relation containing (t, u) if $t = u$ is an instance of an axiom in $\mathrm{BPA}^{\theta}_{\delta\epsilon\tau}$, and as by theorem 8.11 $\leftrightarrows_{P_\theta}$ is also a congruence relation, it is sufficient to show that

$$t = u \text{ is an instance of an axiom in } \mathrm{BPA}^{\theta}_{\delta\epsilon\tau} \;\Rightarrow\; t \leftrightarrows_{P_\theta} u.$$

Suppose $t = u$ is an instance of an axiom. We will only consider axiom P3. All other axioms can be dealt with in the same way. Hence, $t \equiv at' \triangleleft \tau u'$, $u \equiv at' \triangleleft u'$ ($t', u' \in T(\Sigma_\theta)$) and $\neg(a < \tau)$. In order to show that $at' \triangleleft \tau u' \leftrightarrows_{P_\theta} at' \triangleleft u'$, it suffices to show that if $at' \triangleleft \tau u'\xrightarrow{b}v$, ($b \in A_\theta$) then $at' \triangleleft u'\xrightarrow{b}v$ and vice versa, $at' \triangleleft u'\xrightarrow{b}v$ implies $at' \triangleleft \tau u'\xrightarrow{b}v$. So suppose $at' \triangleleft \tau u'\xrightarrow{b}v$. By the structuring lemma one of the following cases must hold:

1. $at'\xrightarrow{b}v$, $b \not\equiv \surd$ and $\forall c > b\ \tau u'\overset{c}{\not\rightarrow}$,
2. $at'\xrightarrow{\tau}t''$, $t''\xrightarrow{b}v$ and $b \not\equiv \surd$ for some $t'' \in T(\Sigma_\theta)$,
3. $at'\xrightarrow{\surd}v$ and $b \equiv \surd$.

Note that case 3 is impossible. So either case 1 or case 2 must hold. If case 2 holds, it is immediately clear that $at' \lhd u' \xrightarrow{\tau} t''$ and $t'' \xrightarrow{b} v$. Therefore, $at' \lhd u' \xrightarrow{b} v$. If case 1 holds, then $\forall c > b\ u' \overset{c}{\not\to}$. If this were not the case, i.e. $\exists c > b\ u' \xrightarrow{c} u''$, then $\tau u' \xrightarrow{c} u''$ contradicting that $\forall c > b\ \tau u' \overset{c}{\not\to}$. Hence $at' \lhd u' \xrightarrow{b} v$.

The other implication can be proved likewise. □

We now show completeness of the axioms. This is done in three stages. First the class of basic terms is introduced. This class is a subset of all closed Σ_θ-terms, but it is still powerful enough to denote all recursion free processes. This is in fact shown in lemma 10.6.

Then operational characteristics are linked to the syntactic forms of terms using the operational soundness and completeness lemmas. In the last lemma all results are gathered together and completeness is shown.

Definition 10.4. The set of *basic terms* is the smallest subset of $T(\Sigma_\theta)$ satisfying:

- δ and ϵ are basic terms,
- if t is a basic term, then at ($a \in Act_\tau$) is a basic term,
- if t, t' are basic terms, then $t + t'$ is a basic term.

Note that $a\epsilon$ and $a\epsilon + b\delta$ are basic terms but a and $(a+b)c$ are not.

Lemma 10.5. *Let t, t' be basic terms. Then there is a basic term u such that:*

1. $\mathrm{BPA}^{\theta}_{\delta\epsilon\tau} \vdash t \Box t' = u$ ($\Box = +, \cdot, \lhd$),
2. $\mathrm{BPA}^{\theta}_{\delta\epsilon\tau} \vdash \Box(t) = u$ ($\Box = \tau_I, \theta$).

Proof. As an example we show the proof for $\lhd$. For a basic term t define $\#t$ as the number of function names in t. Define the depth of a term $t \lhd t'$ with t, t' basic terms by (ω is the first infinite ordinal):

$$D(t \lhd t') = \omega \cdot \#t' + \#t.$$

We prove this case with induction on $D(t \lhd t')$. Distinguish the following cases:

$t = \epsilon, \delta$	Apply PE1 or PD1.
$t' = \epsilon, \delta$	Apply PE2 or PD2.
$t = au_1, t' = bu_2, (b \not\equiv \tau)$	Apply P1 or P2.
$t = au_1, t' = \tau u_2$	Apply P1 if $a < \tau$. If $\neg(a < \tau)$ then $\mathrm{BPA}^{\theta}_{\delta\epsilon\tau} \vdash au_1 \lhd \tau u_2 \overset{P3}{=} au_1 \lhd u_2$. As $D(au_1 \lhd u_2) < D(au_1 \lhd \tau u_2)$, it follows with induction that $\mathrm{BPA}^{\theta}_{\delta\epsilon\tau} \vdash au_1 \lhd u_2 = v$ for some basic term v.
$t' = u_1 + u_2$	We have that $t \lhd t' \equiv t \lhd (u_1 + u_2) \overset{P4}{=} (t \lhd u_1) \lhd u_2$. As $D(t \lhd u_1) < D(t \lhd (u_1 + u_2))$ it follows that $\mathrm{BPA}^{\theta}_{\delta\epsilon\tau} \vdash t \lhd u_1 = v$ for some basic term v. As $D(v \lhd u_2) < D(t \lhd (u_1 + u_2))$ it follows that $\mathrm{BPA}^{\theta}_{\delta\epsilon\tau} \vdash (t \lhd u_1) \lhd u_2 = v \lhd u_2 = v'$ for some basic term v'.
$t = u_1 + u_2$	It follows that $t \lhd t' \equiv (u_1 + u_2) \lhd t' \overset{P5}{=} u_1 \lhd t' + u_2 \lhd t'$. As $D(u_1 \lhd t') < D((u_1 + u_2) \lhd t')$ and $D(u_2 \lhd t') < D((u_1 + u_2) \lhd t')$, there are basic terms v, v' such that $\mathrm{BPA}^{\theta}_{\delta\epsilon\tau} \vdash u_1 \lhd t' = v$ and $\mathrm{BPA}^{\theta}_{\delta\epsilon\tau} \vdash u_2 \lhd t' = v'$. Hence, $\mathrm{BPA}^{\theta}_{\delta\epsilon\tau} \vdash t \lhd t' = v + v'$.

□

Lemma 10.6. *Let $t \in T(\Sigma_\theta)$ be a recursion free term. Then there is a basic term u such that:*

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

Proof. Apply induction on the structure of t. If $t \equiv \epsilon, \delta, a (\in Act_\tau)$ then the basic terms are respectively: ϵ, δ and $a\epsilon$. If $t \equiv t_1 \Box t_2$ ($\Box \equiv +, \cdot, \triangleleft$), it follows with induction that t_1 and t_2 are provably equal to basic terms u_1, u_2. Then lemma 10.5 yields $\mathrm{BPA}^{\theta}_{\delta\epsilon\tau} \vdash u_1 \Box u_2 = u$ with u a basic term. For the unary operators θ and τ_I a similar argument can be applied. $\Box$

The following notation is an abbreviation that turns out to be useful.

Notation 10.7 *(Summand inclusion).* We write $t \subseteq t'$ for $t + t' = t'$.

The following lemmas relate summand inclusion to the operational rules in table 1. They state that if a process t can perform an a-step ($t \xrightarrow{a} t'$) then it is provable that at' is a summand of t. A weak variant of the converse also holds.

Lemma 10.8 *(Operational soundness). Let $t, t' \in T(\Sigma_\theta)$ be recursion free terms and let $a \in Act_\tau$:*

$$\mathrm{BPA}^{\theta}_{\delta\epsilon\tau} \vdash a \cdot t' \subseteq t \;\Rightarrow\; \exists t'' : t \xrightarrow{a} t'' \text{ and } t'' \leftrightarrow_{P_\theta} t',$$
$$\mathrm{BPA}^{\theta}_{\delta\epsilon\tau} \vdash \epsilon \subseteq t \;\Rightarrow\; \exists t' : t \xrightarrow{\surd} t'.$$

Proof. Directly using the soundness theorem 10.3. $\Box$

Lemma 10.9 *(Operational completeness). Let $t, t' \in T(\Sigma_\theta)$ be recursion-free and $\theta, \triangleleft$-free terms and let $a \in Act_\tau$:*

$$t \xrightarrow{a} t' \;\Rightarrow\; \mathrm{BPA}^{\theta}_{\delta\epsilon\tau} \vdash at' \subseteq t,$$
$$t \xrightarrow{\surd} t' \;\Rightarrow\; \mathrm{BPA}^{\theta}_{\delta\epsilon\tau} \vdash \epsilon \subseteq t.$$

Proof. Straightforward induction on the proof of $t \xrightarrow{a} t'$ and $t \xrightarrow{\surd} t'$ from $Strip(P_\theta, \longrightarrow_{P_\theta})$. $\Box$

Lemma 10.10. *Let t be a basic term. If $t \xrightarrow{a} t'$ ($a \in Act_\tau$), then $t' \equiv \epsilon \cdot u$ or $t' \equiv \tau \cdot u$ for some basic term u. Moreover, t' contains at most as many function names as t.*
Proof. Use induction on the proof of $t \xrightarrow{a} t'$ from $Strip(P_\theta, \longrightarrow_{P_\theta})$. $\Box$

Notation 10.11. Let $t, u \in T(\Sigma_\theta)$ be recursion free. $t \rightleftarrows_{P_\theta} u$ stands for: $t \xrightarrow{a} t'$ implies $\exists u'\; u \xrightarrow{a} u'$ and $t' \leftrightarrow_{P_\theta} u'$. Note that this condition resembles clause 1 in the definition of bisimulation.

Lemma 10.12. *Let t and u be basic terms over $\mathrm{BPA}^{\theta}_{\delta\epsilon\tau}$. Then:*

1. *If $t \rightleftarrows_{P_\theta} u$ then $\mathrm{BPA}^{\theta}_{\delta\epsilon\tau} \vdash t \subseteq u$,*
2. *If $t \leftrightarrow_{P_\theta} u$ then $\mathrm{BPA}^{\theta}_{\delta\epsilon\tau} \vdash t = u$.*

Proof. We use induction on the number of function names in t and u, i.e. $\#t + \#u$. The proof employs the operational soundness and completeness lemmas.
Basis. First 1 is proved. Suppose that $t \equiv \epsilon$ and $u \in T(\Sigma_\theta)$. $\epsilon \rightleftarrows_{P_\theta} u \Rightarrow u \xrightarrow{\surd} u' \Rightarrow \mathrm{BPA}^{\theta}_{\delta\epsilon\tau} \vdash \epsilon \subseteq u$. Suppose $t \equiv \delta$. This case is trivial using axiom A6. In case 2 $t \leftrightarrow_{P_\theta} u$ implies $t \rightleftarrows_{P_\theta} u$ and $u \rightleftarrows_{P_\theta} t$, so it follows by 1 that $\mathrm{BPA}^{\theta}_{\delta\epsilon\tau} \vdash t \subseteq u$ and $\mathrm{BPA}^{\theta}_{\delta\epsilon\tau} \vdash u \subseteq t$. Hence $\mathrm{BPA}^{\theta}_{\delta\epsilon\tau} \vdash t = t + u = u + t = u$.
Induction. First consider 1. Suppose $t \equiv (t_1 + t_2) \rightleftarrows_{P_\theta} u$. This implies that $t_1 \rightleftarrows_{P_\theta} u$ and $t_2 \rightleftarrows_{P_\theta} u$. Using 1 inductively yields: $\mathrm{BPA}^{\theta}_{\delta\epsilon\tau} \vdash t_1 \subseteq u$ and $\mathrm{BPA}^{\theta}_{\delta\epsilon\tau} \vdash t_2 \subseteq u$. Now using axiom A1 leads to $\mathrm{BPA}^{\theta}_{\delta\epsilon\tau} \vdash t_1 + t_2 \subseteq u$.

Now suppose that $t \equiv at_1 \rightrightarrows_{P_\theta} u$. Note that $\#t_1 < \#t$. There is a t_2 (e.g. ϵt_1) such that $t \xrightarrow{a} t_2 \leftrightarrows_{P_\theta} t_1$. As $t \rightrightarrows_{P_\theta} u$, it follows that there is a u_1 such that $u \xrightarrow{a} u_1$, $t_1 \leftrightarrows_{P_\theta} t_2 \leftrightarrows_{P_\theta} u_1$. By lemma 10.10 u_1 is a basic term and $\#u_1 \leq \#u$. With the induction hypothesis conclude that $\mathrm{BPA}^{\theta}_{\delta\epsilon\tau} \vdash t_1 = u_1$. By operational completeness it follows that $\mathrm{BPA}^{\theta}_{\delta\epsilon\tau} \vdash au_1 \subseteq u$. Therefore, $\mathrm{BPA}^{\theta}_{\delta\epsilon\tau} \vdash at_1 \subseteq u$.

In case 2 $t \leftrightarrows_{P_\theta} u$ implies $t \rightrightarrows_{P_\theta} u$ and $u \rightrightarrows_{P_\theta} t$, so it follows by 1 that $\mathrm{BPA}^{\theta}_{\delta\epsilon\tau} \vdash t \subseteq u$ and $\mathrm{BPA}^{\theta}_{\delta\epsilon\tau} \vdash u \subseteq t$. Hence $\mathrm{BPA}^{\theta}_{\delta\epsilon\tau} \vdash t = t + u = u + t = u$. □

Theorem 10.13 *(Completeness of* $\mathrm{BPA}^{\theta}_{\delta\epsilon\tau}$*). Let $t, u \in T(\Sigma_\theta)$ be recursion free. It holds that:*

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

Proof. Suppose $t \leftrightarrows_{P_\theta} u$. Then there are basic terms t' and u' that are provably equivalent to t and u. With soundness it follows that $t' \leftrightarrows_{P_\theta} u'$. An application of lemma 10.12 yields $\mathrm{BPA}^{\theta}_{\delta\epsilon\tau} \vdash t' = u'$ and thus $\mathrm{BPA}^{\theta}_{\delta\epsilon\tau} \vdash t = u$. □

Appendix. The relation between TSS's and logic programs

Throughout this paper techniques from logic programming are applied to TSS's. This raises the question whether TSS's can be viewed as logic programs. It appears that there exists indeed a straightforward translation from TSS's to logic programs.

Definition A.1. Let $P = (\Sigma, A, R)$ be a TSS. We define the translation $\mathcal{L}$ as:

for every positive literal $t \xrightarrow{a} t'$, $\mathcal{L}(t \xrightarrow{a} t') = \mathit{transition}(t, a, t')$,
for every negative literal $t \stackrel{a}{\not\rightarrow}$, $\mathcal{L}(t \stackrel{a}{\not\rightarrow}) = \neg \mathit{possible}(t, a)$,
for every rule $r \in R$: $\mathcal{L}(r) = \mathcal{L}(\mathit{conc}(r)) \leftarrow \mathcal{L}(\mathit{prem}(r))^1$,

and finally

$$\mathcal{L}(P) = \mathcal{L}(R)^1 \cup \{\mathit{possible}(T, A) \leftarrow \mathit{transition}(T, A, U)\}$$

where T, A and U are variables.

For an introduction in logic programming we refer to [23]. We must point out some small differences between the two formalisms.

First of all, logic programs are usually untyped, whereas a TSS $P = (\Sigma, A, R)$ has clearly two types, namely terms (from $T(\Sigma)$) and labels (from A). Thus the translation $\mathcal{L}(P)$ must also be treated as a typed program, its Herbrand base being

$$\begin{aligned} HB_P &= \{\mathit{transition}(t, a, t') | t, t' \in T(\Sigma),\ a \in A\} \cup \\ &\quad\ \{\mathit{possible}(t, a) | t \in T(\Sigma),\ a \in A\}. \end{aligned}$$

Secondly, a traditional logic program consists of a finite set of finite clauses. A TSS may have an infinite number of rules and each rule may have infinitely many premises. The main reason for this is that in TSS's only variables ranging over terms are used, and no variables ranging over labels. Thus instead of one rule like

$$\frac{x \xrightarrow{z} x'}{x + y \xrightarrow{z} x'},$$

this rule must be incorporated for every action z separately. Usually rule schemes with meta-variables ranging over A are given, like in this case rule R3.1 of the running example. Translating a TSS yields a possibly infinite set of possibly infinite clauses. Of course having an infinite number of clauses is not

^{1}as usual $\mathcal{L}(X)$ abbreviates $\{\mathcal{L}(x) | x \in X\}$

a problem: the set of ground instances of clauses from a traditional logic program is normally infinite as well. Having infinitely many premises seems harmless too.

In order to formulate the intended correspondencies between the TSS P and the logic program $\mathcal{L}(P)$, we also need a translation on the semantical level, i.e. between transitions relations and (well-typed) Herbrand interpretations.

Definition A.2. Let $\longrightarrow$ be a transition relation.

$$\begin{array}{rcl}\mathcal{M}(\longrightarrow) & = & \{transition(t,a,t') \mid t \xrightarrow{a} t' \in \longrightarrow\} \cup \\ & & \{possible(t,a) \mid \exists t' : t \xrightarrow{a} t' \in \longrightarrow\}.\end{array}$$

According to this definition only interpretations M satisfying for all t and a:

$$possible(t,a) \in M \quad \text{iff} \quad \exists t' : transition(t,a,t') \in M$$

are translations of a transition relation. The clause $possible(T,A) \leftarrow transition(T,A,U)$ is obviously incorporated in the translation of every TSS to enforce this property. As long as only supported models of the resulting logic programs are considered, the addition of this clause in indeed sufficient. The following example shows that a weaker choice of semantics (in this case minimal models) can produce certain anomalous models.

Example A.3. Consider the TSS P with one constant c and one unary function f, one action a and the following rules:

$$\frac{c \overset{a}{\not\longrightarrow}}{c \xrightarrow{a} c} \qquad\qquad \frac{c \xrightarrow{a} x}{c \xrightarrow{a} f(x)}.$$

For all $n \geq 0$, the transition relation $\{c \xrightarrow{a} f^i(c) \mid i \geq n\}$ is a model of P; P has no other models. As n increases, the model decreases (w.r.t. $\subseteq$), thus P has no minimal model. Now consider

$$\begin{array}{rcl}\mathcal{L}(P) & = & \{transition(c,a,c) \leftarrow \neg possible(c,a) \\ & & transition(c,a,f(X)) \leftarrow transition(c,a,X) \\ & & possible(T,A) \leftarrow transition(T,A,U)\}.\end{array}$$

The corresponding models are (for all $n \geq 0$):

$$\{transition(c,a,f^i(c)) \mid i \geq n\} \cup \{possible(c,a)\}.$$

But $\mathcal{L}(P)$ has one more model, namely just $\{possible(c,a)\}$, which is the least model of $\mathcal{L}(P)$, but not supported by $\mathcal{L}(P)$.

As we concentrate on the stable and well-founded model semantics, which generate only supported models, anomalous models will no more arise.

In the rest of this section we establish the relationships between TSS's and their translations into logic programs. For the definitions regarding logic programming we refer to the literature. As these definitions are always similar to the definitions regarding TSS's as presented in this paper, it is straightforward to prove the following propositions.

Proposition A.4. *Let P be a TSS.*

- *P is positive iff $\mathcal{L}(P)$ is positive.*
- *P is stratified iff $\mathcal{L}(P)$ is locally stratified (see [30]).*

$$\begin{tikzcd}
P \text{ (a TSS)} \arrow[r, leftrightarrow, "\mathcal{L}"] \arrow[d, "\text{associated transition relation}"'] & \mathcal{L}(P) \text{ (a logic program)} \arrow[d, "\text{associated Herbrand interpretation}"] \\
\longrightarrow_P \arrow[r, leftrightarrow, "\mathcal{M}"] & \mathcal{M}(\longrightarrow_P)
\end{tikzcd}$$

Figure 2: The relation between TSS's and logic programs

For a positive logic program P, M_P denotes its least Herbrand model.

Proposition A.5. *Let P be a TSS and $\longrightarrow$ be a transition relation.*
$\longrightarrow$ is stable for P iff $\mathcal{M}(\longrightarrow)$ is a stable model of $\mathcal{L}(P)$ (see [13]).
In particular, if P is positive, then $\mathcal{M}(\longrightarrow_P) = M_{\mathcal{L}(P)}$ and
if P is stratified, then $\mathcal{M}(\longrightarrow_P)$ is the unique perfect model of $\mathcal{L}(P)$ (see [30]).

Two slightly different, but equivalent, definitions of well-founded models for logic programs have been given ([12] and W_P in [33] versus [31] and V_P in [33]). Here we follow [31].

Definition A.6 *(well-founded model).* Let P be a logic program.

- A 3-valued interpretation for P is a pair $I =< T, F >$, where T and F are subsets of the Herbrand base HB_P (but not necessarily $T \cap F = \emptyset$).
- Let A be a ground atom. Then $< T, F > \models A$ iff $A \in T$ and $< T, F > \models \neg A$ iff $A \in F$.
- $T_P(I) = \{A \in HB_P |$ there exists a clause $A \leftarrow L_1, ..., L_n \in ground(P)$ such that $I \models L_1$ and ... and $I \models L_n\}$.

  $F_P(I) = \{A \in HB_P |$ for every clause $A \leftarrow L_1, ..., L_n \in ground(P)$: $I \models \neg L_1$ or ... or $I \models \neg L_n\}$.
  (If L is a negative literal $\neg B$, then $\neg L$ denotes B.)

  $T_P(I)$ defines the ground atoms that are immediately true given P and I, $F_P(I)$ defines the ground atoms that are immediately false.
- Let $T, F \subseteq HB_P$ and let I be a 3-valued interpretation for P.

  $\mathcal{T}_I(T) = T_P(I \cup < T, \emptyset >)$.
  $\mathcal{F}_I(F) = F_P(I \cup < \emptyset, F >)$.
  $\mathcal{I}_P(I) = I \cup < \bigcup_{n<\omega} \mathcal{T}_I^n(\emptyset), \bigcap_{n<\omega} \mathcal{F}_I^n(HB_P) >$. (Note: $\cup$ denotes pointwise union.)

  $\mathcal{I}_P(I)$ defines the ground atoms that are certainly true respectively false given P and I.
- For a limit ordinal α: $I_\alpha = \bigcup_{\beta<\alpha} I_\beta$ (in particular: $I_0 =< \emptyset, \emptyset >$).
  For a successor ordinal $\alpha + 1$: $I_{\alpha+1} = \mathcal{I}_P(I_\alpha)$.
- Let δ be the smallest countable ordinal such that $I_\delta = \mathcal{I}(I_\delta)$. Then I_δ is the well-founded (partial) model of P. If I_δ is 2-valued, i.e. $I_\delta =< T, F >$ is a partitioning of HB_P, then I_δ is the well-founded (complete) model of P.

An alternative definition of the well-founded model, based on the reduction of logic programs, can also be given.

Definition A.7. Let P be a logic program and I a 3-valued interpretation for P. Then:

$$Reduce(P,I) = \bigcup_{C \in ground(P)} Reduce(C,I), \text{ where}$$
$$Reduce(A \leftarrow S, I) = \begin{cases} \emptyset & \text{if for some literal } L \in S: \ I \models \neg L \\ \{A \leftarrow S'\} & \text{otherwise, where } S' = \{L \in S | I \not\models L\}. \end{cases}$$

Furthermore:

$$True(P) = \{A \leftarrow S \in P | S \text{ contains only positive literals}\} \text{ and}$$
$$Pos(P) = \{A \leftarrow S' | \text{there is a clause } A \leftarrow S \in P \text{ such that } S' \text{ is the set of positive literals in } S\}.$$

Lemma A.8. *Let P be a logic program and I a 3-valued interpretation for P.*

$$\bigcup_{n<\omega} \mathcal{T}_I^n(\emptyset) = M_{True(Reduce(P,I))},$$
$$\bigcap_{n<\omega} \mathcal{F}_I^n(HB_P) = HB_P - M_{Pos(Reduce(P,I))}.$$

Thus an alternative definition of the well-founded (partial) model of a logic program is obtained by replacing

$$\bigcup_{n<\omega} \mathcal{T}_I^n(\emptyset) \text{ by } M_{True(Reduce(P,I))} \text{ and}$$
$$\bigcap_{n<\omega} \mathcal{F}_I^n(HB_P) \text{ by } HB_P - M_{Pos(Reduce(P,I))}$$

in definition A.6. The proof of lemma A.8 is beyond the scope of this paper.

Using this alternative definition, it is straightforward to link the reduction of a TSS P and the sequence of interpretations leading to the well-founded (partial) model of $\mathcal{L}(P)$.

Lemma A.9. *Let $P = (\Sigma, A, R)$ be a TSS and let $\longrightarrow_{true}, \longrightarrow_{pos} \subseteq Tr(\Sigma, A)$. Then:*

- $\mathcal{L}(True(P)) = True(\mathcal{L}(P))$,
- $\mathcal{L}(Pos(P)) = Pos(\mathcal{L}(P))$,
- $\mathcal{L}(Reduce(P, \longrightarrow_{true}, \longrightarrow_{pos})) = Reduce(\mathcal{L}(P), < \mathcal{M}(\longrightarrow_{true}), \mathrm{HB}_P - \mathcal{M}(\longrightarrow_{pos}) >)$.

Theorem A.10. *Let P be a TSS. Let for all ordinals α, I_α be defined w.r.t. $\mathcal{L}(P)$ as in definition A.6. Then:*

$$\mathcal{L}(Red^\alpha(P)) = Reduce(\mathcal{L}(P), I_\alpha),$$

$$I_\alpha = < \bigcup_{\beta<\alpha} \mathcal{M}(\longrightarrow_{True(Red^\beta(P))}), \bigcup_{\beta<\alpha} \mathrm{HB}_P - \mathcal{M}(\longrightarrow_{Pos(Red^\beta(P))}) > .$$

Proof. Straightforward. $\square$

Corollary A.11. *Let P be a TSS. If $\mathcal{L}(P)$ has a well-founded complete model I_α, then $Red^\alpha(P)$ is a positive TSS and $I_\alpha = \mathcal{M}(\longrightarrow_{Red^\alpha(P)})$. If $Red^\alpha(P)$ is a positive TSS then $\mathcal{L}(P)$ has a well-founded complete model $I_{\alpha+1} = \mathcal{M}(\longrightarrow_{Red^\alpha(P)})$.*

The change from α to $\alpha + 1$ in the second implication is caused by the fact that it is possible that at the end of the iteration first the interpretation I_α becomes 2-valued (making *Reduce*$(\mathcal{L}(P), I_\alpha)$ positive), but also that a partial I_α results in a positive *Reduce*$(\mathcal{L}(P), I_\alpha)$, in which case only $I_{\alpha+1}$ is 2-valued.

Apart from its theoretical merits, the translation of TSS's into logic programs has also more practical implications. For logic programs interpreters and compilers are available. Thus in order to find out whether a term t can perform an a-step according to $\longrightarrow_P$, the TSS P is translated into the logic program $\mathcal{L}(P)$, and the query $\leftarrow$ *transition*(t, a, X) is presented to it.

For positive TSS's this poses only one problem: the depth-first strategy of most programming systems tends to result in non-termination of the program without finding all solutions. The rules R9.2 and R9.3 of the running example are typically rules leading to non-termination. Incorporating certain forms of loop-checking ([7, 36]) might partly solve the problem, but as even for positive TSS's $\longrightarrow_P$ need not be recursive, non-termination can never be ruled out completely.

In the presence of negation $\longrightarrow_P$ is in general not even recursive enumerable and the execution of the logic program becomes even more involved (but see [31, 33], where 'ideal' mechanisms are proposed for computing the well-founded model, abstracting away from non-termination). Thus the translation into logic programming cannot be expected to produce the associated transition relation as a whole. But in our opinion the interactive use of a logic programming environment for proving that a certain transition holds (or does not hold) is an attractive alternative to generating this proof by hand, especially for larger TSS's.

On the bright side is that for pure TSS's (see definition 8.3) the problem of *floundering* (the necessity to resolve a non-ground negative literal, see e.g. [23]) does not occur for queries of the form $\leftarrow$ *transition*(t, a, X) (with $t \in T(\Sigma)$ and $a \in A$). This can be shown by *annotating* the program (in the sense of [11]) by *transition*$(\downarrow, \downarrow, \uparrow)$ and *possible*$(\downarrow, \downarrow)$, meaning that the first and second argument of both predicates are considered to be input, and the third argument of *transition* is output. (Due to the fact that TSS's have no variables ranging over labels, the annotations of the second (label) arguments are inessential.)

Proposition A.12. *Let $P = (\Sigma, A, R)$ be a TSS. Let $t \in T(\Sigma)$ and $a \in A$. If P is pure then $\mathcal{L}(P) \cup \{\leftarrow transition(t, a, X)\}$ is well-formed (see [11]) w.r.t. the above annotation.*

The well-formedness of a logic program and a query implies that during the computation every predicate is called with ground terms on its input arguments. In particular, every call $\neg$*possible*(t, a) will be ground. In a more general setting, this annotation gives insight in the data-flow of the act of proving transitions from pure TSS's.

References

[1] J.C.M. Baeten and J.A. Bergstra. Processen en procesexpressies. *Informatie*, 30(3):177–248, 1988. In Dutch.

[2] 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.

[3] E. Best and M. Koutny. Partial order semantics of priority systems. Technical Report 6/90, Universität Hildesheim, Institut für Informatik, 1990.

[4] N. Bidoit and C. Froidevaux. Minimalism subsumes default logic and circumscription. In *Proceedings IEEE Conference on Logic in Computer Science* (LICS87), New York, pages 89–97, 1987.

[5] N. Bidoit and C. Froidevaux. Negation by default and non stratifiable logic programs. Technical Report 437, L.R.I. France, 1988.

[6] 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.

[7] R.N. Bol, K.R. Apt, and J.W. Klop. An analysis of loop checking mechanisms for logic programs. Technical Report CS-R8947, CWI, Amsterdam, 1989. To appear in *Theoretical Computer Science.*

[8] 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.

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

[10] 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.

[11] P. Dembinski and J. Małuszynski. And-parallelism with intelligent backtracking for annotated logic programs. In *Symposion on Logic Programming,* Boston, pages 29–38, 1985.

[12] 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.

[13] 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.

[14] 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.

[15] R.J. van Glabbeek. The linear time - branching time spectrum. In J.C.M. Baeten and J.W. Klop, editors, *Proceedings Concur90,* Amsterdam, LNCS 458, pages 278–297. Springer Verlag, 1990.

[16] 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.

[17] 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.

[18] M. Hennessy and G.D. Plotkin. Full abstraction for a simple programming language. In J. Bečvář, editor, *Proceedings Eigth Symposium on Mathematical Foundations of Computer Science* (MFCS), LNCS 74, pages 108–120. Springer-Verlag, 1979.

[19] M. Hennessy and T. Regan. A temporal process algebra. Technical Report 2/90, Computer Science Department, University of Sussex, 1990.

[20] H. Ichikawa, K. Yamanaka, and J. Kato. Incremental specifications in 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.

[21] R. Janicki. A formal semantics for concurrent systems with a priority relation. *Acta Informaticae,* 24:33–55, 1987.

[22] R. Langerak. A testing theory for LOTOS using deadlock detection. In E. Brinksma, G. Scollo, and C.A. Vissers, editors, *Proceedings Ninth IFIP WG6.1 International Symposium on Protocol Specification, Testing, and Verification,* Enschede, 1989.

[23] J.W. Lloyd. *Foundations of Logic Programming, Second Edition.* Springer-Verlag, 1987.

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

[25] X. Nicollin, J.-L. Richier, J. Sifakis, and J. Voiron. ATP: An algebra for timed processes. Technical Report RT-C16, IMAG, Laboratoire de Génie informatique, Grenoble, 1990. This article also appeared in M. Broy and C.B. Jones, editors, *Proceedings IFIP Working Conference on Programming Concepts and Methods*, Sea of Gallilea, Israel. North Holland, 1990.

[26] 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.

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

[28] 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.

[29] H. Przymusinska and T. Przymusinski. Weakly perfect model semantics for logic programs. In R. Kowalski and K. Bowen, editors, *Proceedings of the Fifth Logic Programming Symposium,* pages 1106–1120. MIT press, 1988.

[30] 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.

[31] T.C. Przymusinski. Every logic program has a natural stratification and an iterated least fixed point model. In *Proceedings of the Eighth Symposium on Principles of Database Systems,* pages 11–21. ACM SIGACT-SIGMOD, 1989.

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

[33] K. Ross. A procedural semantics for well founded negation in logic programs. In *Proceedings of the Eighth Symposium on Principles of Database Systems,* pages 22–33. ACM SIGACT-SIGMOD, 1989.

[34] F.W. Vaandrager. Specificatie en verificatie van communicatieprotocollen met procesalgebra. Unpublished, in Dutch.

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

[36] L. Vieille. Recursive query processing: the power of logic. *Theoretical Computer Science,* 69(1):1–53, 1989.