A Conservative Look at Operational Semantics with Variable Binding

Wan Fokkink
University of Wales Swansea
Department of Computer Science
Singleton Park, Swansea SA2 8PP, Wales
w.j.fokkink@swan.ac.uk

Chris Verhoef
University of Amsterdam
Department of Computer Science
Programming Research Group
Kruislaan 403, 1098 SJ Amsterdam, The Netherlands
x@wins.uva.nl

Abstract

We set up a formal framework to describe transition system specifications in the style of Plotkin. This framework has the power to express many-sortedness, general binding mechanisms and substitutions, among other notions such as negative hypotheses and unary predicates on terms.

The framework is used to present a conservativity format in operational semantics, which states sufficient criteria to ensure that the extension of a transition system specification with new transition rules does not affect the semantics of the original terms.

1 Introduction

A current method to provide process algebras and specification languages with an operational semantics is based on the use of structured operational semantics from Plotkin [44]. Given a set of states, the transitions between these states are obtained inductively from a transition system specification (TSS), which consists of transition rules.

Desirable properties for the transition systems that are generated by some TSS are often deduced by means of long technical proofs. Therefore, several general formats for TSSs have been developed, for instance to determine which TSSs satisfy a certain congruence property [49, 12, 32, 29, 14, 6, 53, 20, 38, 11], or to study the meaning of negative hypotheses [29, 14, 28], or to find which extensions of TSSs are operationally conservative [32, 29, 14, 52, 18, 19]. Our article is devoted to this last topic.

Over and over again, process theories such as CCS [39], CSP [37] and ACP [8] have been extended with new features, and the original TSSs, which provide the semantics for these process algebras, were extended with transition rules to describe these features; see [7] for a systematic approach. A question that arises naturally is whether or not such an extension influences the transition systems of terms in the original domain. Usually, it is desirable that an extension is (operationally) conservative, meaning that the provable transitions for an original term are the same both in the original and in the extended TSS.

Groote and Vaandrager [32, Theorem 7.6] proposed the first syntactic restrictions for an original TSS and its extension, which automatically yield that the extension is operationally conservative. The restrictions are: all transition rules must be 'tyft/tyxt', and the original transition rules must be 'pure' and 'well-founded' (see [32] for the definitions), and the transition rules in the extension must contain some fresh operator in their source, i.e., in the left-hand side of their conclusion. Groote [29] adapted this conservativity format to the setting with negative hypotheses. Bol and Groote [14] showed that the tyft/tyxt restriction can be omitted.

Verhoef [52] proposed more general syntactic criteria which ensure operational conservativity. Verhoef's criteria allow, under certain conditions, that a transition rule in the extension has an original term as its source. Examples of extensions that are within the scope of Verhoef's criteria, but that do not fit the previous formats, are the extension of CCS with time from Moller and Tofts [42], and BPA with discrete time from Baeten and Bergstra [4]. (In a later version of BPA with discrete time [5], the operational semantics has been adapted in such a way that the extension with discrete time is no longer operationally conservative over BPA.) Verhoef's format was extended to a setting with inequalities in [18, 19].

In many practical cases, the format from [52] cannot yet be applied, due to the use of a many-sorted signature, or the presence of some variable binding mechanism in the transition rules. Familiar examples of such binding mechanisms are the expression $\lambda x.t$ from the λ-calculus, where the variable x is bound in the term t, and the construct $t[s/x]$, where occurrences of the variable x in the term t are replaced by the term s. This article proposes a generalization of the conservativity format from [52] to transition rules which may contain many-sortedness and a variable binding mechanism. This generalization requires a subtle distinction between several kinds of occurrences of variables in transition rules. We relax the criteria 'pure' and 'well-founded', that were posed on the original transition rules by all previous conservativity formats, to a more natural requirement on variables in the original transition rules, which we call 'source-dependency'. Furthermore, variables in original transition rules need not be source-dependent, under the condition that the sorts of such variables are not extended with fresh terms. Finally, we allow terms as labels in transition rules; Bernstein [10] showed that such labels allow to capture higher-order languages.

Several concepts in the setting of operational semantics with variable binding, which seem to be intuitively clear at first sight, turn out to be ambiguous when studied carefully. In order to obtain a formal framework in which transition rules with a variable binding mechanism can be expressed rigorously, we elaborately discuss the preliminaries, presenting examples and introducing new notions on the way. Most notably, we distinguish between actual and formal variables, following conventions from programming languages, and we formalize the construct $t[s/x]$ in transition rules.

We give two detailed examples to show how our conservativity format can be applied to practical cases. The examples deal with real time ACP [21] and the πI-calculus [47].

A check on the source-dependency of transition rules has been incorporated in the tool LATOS [33]. In [23] part of the conservative extension format presented in this article has been transposed to positive/negative conditional term rewriting systems, and shown to be applicable with respect to software renovation factories.

Acknowledgements. We have benefited from technical discussions with Bard Bloom, Pedro D'Argenio, Anuj Dawar, Arie van Deursen, Rob van Glabbeek, Pieter Hartel, Douglas Howe, Davide Sangiorgi, and Frits Vaandrager. An anonymous referee provided valuable comments.

2 The Formal Framework

In this section we recall some notions concerning general theory of structured operational semantics, and introduce some new matters, interspersed with examples. We define a framework in which it is possible to express binding mechanisms and substitutions, and incorporate the notions of negative hypotheses from Groote [29] and predicates from Baeten and Verhoef [6]. Furthermore, we introduce two different kinds of terms: actual ones and formal ones.

Some intuitions In many programming languages there are so-called actual parameters and formal parameters. The formal parameters are used to define procedures or functions; the actual parameters are the "real" variables to be used in the main program. In the main program the formal parameters are bound by the actual parameters. When discussing procedures on a conceptual level, it is often useful to introduce a notational distinction between formal and actual parameters; see for instance [55]. We do the same in this article: we think of a transition rule as a procedure to establish a transition relation by means of substituting (actual) terms for the (formal) variables. Since transition rules are discussed on a conceptual level, we make a clear distinction between actual and formal variables. Transition rules are built from terms that may contain formal variables, and proofs for transitions are obtained by substituting actual terms for formal variables in transition rules.

The following example illustrates that it is useful to make a notational distinction between actual and formal variables. Consider the transition rule

$$\frac{y[w/x]\xrightarrow{a}z}{y\xrightarrow{b}z}$$

where w, x, y, z are variables, and $y[w/x]$ is a standard notation that binds the x in y, and replaces it by w. Application of a substitution σ to this transition rule yields

$$\frac{\sigma(y)[\sigma(w)/x]\xrightarrow{a}\sigma(z)}{\sigma(y)\xrightarrow{b}\sigma(z)}$$

For instance, if $\sigma(w) = c$ and $\sigma(y) = x$ and $\sigma(z) = d$, then we obtain

$$\frac{c\xrightarrow{a}d}{x\xrightarrow{b}d}$$

We make two observations.

1. The expression $y[w/x]$ is not a substitution (for then it would equal y), but a syntactic construct with a suggestive form. We call it a *substitution harness.* Only after application of a substitution σ, the result $\sigma(y)[\sigma(w)/x]$ can be evaluated to a term.

2. Substitutions only apply to part of the variables that occur in a transition rule. In order to distinguish such variables in a transition rule, we call them *formal*, and we mark them with an asterisk (*).

Hence, the transition rule above takes the form

$$\frac{y^*[w^*/x]\xrightarrow{a}z^*}{y^*\xrightarrow{b}z^*}$$

The distinction of formal variables in structured operational semantics with variable binding was also propagated independently by Sangiorgi [46] and Howe [38]. There, they are called 'meta-variables'.

Now that we have an idea of the framework, we first introduce the notion of actual terms (as opposed to formal terms), in which it is possible to express variable binding. Binding mechanisms exist in many and diverse forms. We describe these mechanisms as general as possible, using a notational approach based on [1]; it is the notation for terms in the Nuprl proof development system; see [17]. The choice for the Nuprl notation, instead of for example the λ-calculus [9], is simply a matter of taste.

2.1 The Actual World

In this section we describe the actual world, which contains actual terms, actual substitutions, and so forth. In the sequel, $\vec{O}$ denotes a sequence $O_1...O_k$, and $\vec{O}_i$ a sequence $O_{i1}...O_{ik}$, with $k \geq 0$.

Definition 2.1 *A (many-sorted) signature Σ consists of a set of sorts, an infinite set $\mathcal{V}$ of sorted actual variables, and a set of function symbols*

$$f : \vec{S}_1.S_1 \times \cdots \times \vec{S}_n.S_n \to S,$$

where the S_{ij} and the S_i and S are sorts.

A function symbol of arity zero is called a *constant.*

Definition 2.2 *Let Σ be a signature. The collection $\mathbb{T}(\Sigma)$ of* (open) actual terms $s, t, ...$ *over Σ is defined as the least set satisfying:*

- *each actual variable from $\mathcal{V}$ is in $\mathbb{T}(\Sigma)$,*
- *for each function $f : \vec{S}_1.S_1 \times \cdots \times \vec{S}_n.S_n \to S$, $f(\vec{x}_1.t_1, ..., \vec{x}_n.t_n)$ is an actual term of sort S, where*

- *the actual terms t_i are of sort S_i,*
- *each $\vec{x}_i$ is a sequence of distinct actual variables $x_{i1}...x_{ik_i}$, with x_{ij} of sort S_{ij}.*

The actual variables $\vec{x}_i$ are said to be bound in the ith argument of f.

Definition 2.3 Free *occurrences of actual variables in actual terms are defined as expected:*

- *x occurs* free *in x for each $x \in \mathcal{V}$;*

- *if x occurs* free *in t_i, and x does not occur in the sequence $\vec{x}_i$, then x occurs* free *in $f(\vec{x}_1.t_1, ..., \vec{x}_n.t_n)$.*

An actual term is called closed *if it does not contain any free occurrences of actual variables. In the sequel, $\mathcal{T}(\Sigma)$ denotes the collection of closed actual terms over Σ.*

The notion of a substitution is also defined as expected.

Definition 2.4 *An* actual substitution *is a sort preserving mapping $\sigma : \mathcal{V} \to \mathcal{T}(\Sigma)$, where sort preserving means that x and $\sigma(x)$ are always of the same sort. A substitution extends to a mapping from open actual terms to closed actual terms as usual; the term $\sigma(t)$ is obtained by replacing each free occurrence of an actual variable x in t by $\sigma(x)$.*

As usual, $\_[t/x]$ is the postfix notation for the substitution that maps x to t and is inert otherwise. Such postfix denoted substitutions are called explicit *actual substitutions (as opposed to* implicit *actual substitutions σ).*

In the definition of actual substitutions on open actual terms there is a well-known complication. Namely, consider an actual term $\sigma(t)$, and let x occur free in t. After x in t has been replaced by $\sigma(x)$, actual variables y that occur in $\sigma(x)$ are suddenly bound in actual subterms such as $f(y.s)$ of t. A solution for this problem, which originates from the λ-calculus, is to allow unrestricted substitution by applying α-conversion, that is, by renaming bound actual variables. In the sequel, actual terms are considered modulo α-conversion, and when a substitution is applied, bound actual variables are renamed. Stoughton [50] presented a nice treatment of this technique.

Remark 2.5 Bloom and Vaandrager [13] developed a framework for transition rules with many-sortedness and a binding mechanism. They make a clear distinction between sorts for processes, which exhibit behaviour, and sorts for data, which do not exhibit any behaviour. This distinction is not of interest for the question whether an extension of transition rules influences the behaviour of original terms. We consider data as processes that do not display any behaviour.

2.2 The Formal World

We argued that it is a good idea to distinguish between formal and actual variables, when discussing transition rules with variable bindings and substitutions on an abstract level. We introduce the notion of a formal term t^*, being an actual term with possible occurrences of formal variables and substitution harnesses.

Assume a signature Σ, consisting of a non-empty set of sorts, a set $\mathcal{V}$ of actual variables, and a set of function symbols. The set $\mathcal{V}^*$ of *formal variables* is defined as $\{x^* \mid x \in \mathcal{V}\}$, where x^* and x are of the same sort.

Definition 2.6 *The collection $\mathbb{F}(\Sigma)$ of* formal terms *over a signature Σ is the least set satisfying:*

- *each actual variable from $\mathcal{V}$ is in $\mathbb{F}(\Sigma)$;*
- *each formal variable from $\mathcal{V}^*$ is in $\mathbb{F}(\Sigma)$;*
- *for each function symbol $f : \vec{S}_1.S_1 \times \cdots \times \vec{S}_n.S_n \to S$, $f(\vec{x}_1.t_1^*, ..., \vec{x}_n.t_n^*)$ is a formal term of sort S, where*
  - *the formal terms t_i^* are of sort S_i,*
  - *each $\vec{x}_i$ consists of distinct actual variables in $\mathcal{V}$ of sorts $\vec{S}_i$;*
- *if s^* and t^* are formal terms of sorts S_0 and S_1 respectively, and $x \in \mathcal{V}$ is of sort S_1, then $t^*[s^*/x]$ is a formal term of sort S_0.*

Definition 2.7 *A* formal substitution *is a sort preserving mapping $\sigma^* : \mathcal{V}^* \to \mathbb{T}(\Sigma)$. It extends to a mapping $\sigma^* : \mathbb{F}(\Sigma) \to \mathbb{T}(\Sigma)$ as expected; the term $\sigma^*(t^*)$ is obtained from t^* by replacing each formal variable x^* in t^* by $\sigma^*(x^*)$, after which the substitution harnesses become explicit actual substitutions. The result evaluates to a term in $\mathbb{T}(\Sigma)$.*

Example 2.8 An example of a formal term is $y^*[w^*/x]$, which evaluates to the actual term a after application of a formal substitution σ^* with $\sigma^*(w^*) = c$ and $\sigma^*(y^*) = x$. Namely, the implicit formal substitution σ^* turns the substitution harness $y^*[w^*/x]$ into the actual term $x[c/x]$, where $\_[c/x]$ is an explicit actual substitution, which evaluates to c.

Summarizing the various substitutions At this point we have introduced all the substitutions and the substitution harness. We summarize the various notions, and briefly discuss their differences. There are four notions in two worlds: the implicit and explicit actual substitutions (which are semantically the same), and the formal substitutions and the substitution harnesses.

- Implicit actual substitutions σ and explicit actual substitutions $\_[t/x]$ both denote mappings from actual variables to closed actual terms.
- Formal substitutions σ^* are mappings from formal variables to open actual terms.
- A substitution harness $t^*[s^*/x]$ is *not* a substitution, but a piece of syntax with a suggestive form. If a formal substitution σ^* is applied to it, then the result is an expression $\sigma^*(t^*)[\sigma^*(s^*)/x]$, containing an explicit actual substitution, so that it can be evaluated to an actual term.

Substitution harnesses are used to formulate in a precise way how a formal substitution is to act on a transition rule. The formal and actual substitutions are used to move from transition rules to a proof tree.

2.3 Actual and Formal Transition Rules

We have explained what the formal framework looks like more or less, and the intuition behind the use of structured operational semantics with variable binding and substitution harnesses. We formalize what that intuition is, in order to be able to discuss the theory of structured operational semantics for higher-order languages on an abstract level, and to give a rigorous presentation of a conservativity result.

Before presenting the basic definitions of structured operational semantics, first we consider as an example the well-known recursive μ-construct, which combines formal variables, a binding mechanism and a substitution harness. This transition rule, which occurs for instance in the operational semantics of [43], serves as a running example.

Example 2.9 Intuitively, the term $\mu x.p$ executes p until it encounters an expression x, in which case it starts to execute $\mu x.p$ again. This intuition is expressed in the following transition rule, which we call the μ-rule:

$$\frac{y^*[\mu x.y^*/x] \xrightarrow{a} z^*}{\mu x.y^* \xrightarrow{a} z^*}$$

Recall that formal variables are marked with an asterisk (*) in order to avoid notational confusion. Note that the variable x in the μ-rule does not carry an asterisk, because we want to bind actual variables to actual terms in the end. The transition

$$\mu x.ax \xrightarrow{a} \mu x.ax$$

with $a\_$ the well-known action prefix operator from CCS, can be derived from the μ-rule together with the standard transition rule for the prefix operator: $aw^* \xrightarrow{a} w^*$. Namely, after application of the formal substitution σ^* to the μ-rule with $\sigma^*(y^*) = ax$ and $\sigma^*(z^*) = \mu x.ax$, the hypothesis takes the form $ax[\mu x.ax/x] \xrightarrow{a} \mu x.ax$, which evaluates to $a\mu x.ax \xrightarrow{a} \mu x.ax$. Since this is an instance of the transition rule for the prefix operator, with $\mu x.ax$ for w^*, we conclude that the σ^*-instantiation of the conclusion of the μ-rule is valid: $\mu x.ax \xrightarrow{a} \mu x.ax$.

We introduce the basic notions of structured operational semantics. We assume a signature Σ, and a set $\mathcal{D}$ of *relation* and *predicate* symbols.

Definition 2.10 *Let $t_0, ..., t_n \in \mathcal{T}(\Sigma)$.*

- *For R a relation, the expression $t_0R(t_1, ..., t_{n-1})t_n$ is a* positive transition*.*
- *For R a predicate, the expression $t_0R(t_1, ..., t_{n-1})$ is a* positive transition*.*
- *For R a relation or a predicate, the expression $t_0\neg R(t_1, ..., t_{n-1})$ is a* negative transition*.*

We allow the possibility to attach terms to relations and predicates, because nowadays many formalisms, such as the π-calculus [41], use transition rules with parametrized labels.

Remark 2.11 Our conservativity result would also hold in a setting where transitions, and proofs of such transitions (as is defined in Section 2.4), may involve open terms; see [22]. However, the assumption that terms in transitions are closed is a standard restriction in applications of operational semantics, so that we refrain from a generalization to open terms.

Definition 2.12 *An* actual (transition) rule *is an expression of the form H/τ, where H is a collection of positive and negative transitions, and τ is a positive transition.*

Example 2.13 An example of an actual rule that we met in Example 2.9 is

$$\frac{a\mu x.ax \xrightarrow{a} \mu x.ax}{\mu x.ax \xrightarrow{a} \mu x.ax}$$

It was deduced from the μ-rule, which is an example of a formal rule.

Actual transition rules are deduced by means of *formal* transition rules. The formal rules are the ones that are presented in the literature; they are the recipes that enable to deduce a transition relation.

Definition 2.14 *A* formal (transition) rule *is an expression of the form H^*/τ^*, where:*

- *H^* is a collection* hypotheses *of the form*
  - *$t_0^* R(t_1^*, ..., t_{n-1}^*) t_n^*$ with R a relation, and*
  - *$t_0^* R(t_1^*, ..., t_{n-1}^*)$ with R a predicate, and*
  - *$t_0^* \neg R(t_1^*, ..., t_{n-1}^*)$ with R a relation or predicate;*
- *τ^* is the* conclusion *of the form*
  - *$t_0^* R(t_1^*, ..., t_{n-1}^*) t_n^*$ with R a relation, or*
  - *$t_0^* R(t_1^*, ..., t_{n-1}^*)$ with R a predicate;*

whereby $t_0^*, ..., t_n^* \in \mathbb{F}(\Sigma)$.
A transition system specification *(TSS) is a collection of formal rules.*

We give an intricate example of a formal transition rule PRE from the π-calculus, which incorporates bound variables and parametrized labels. Recall that actual terms are considered modulo α-conversion.

Example 2.15 Assume two sorts *Port* of port names and *Process* of processes. For actual variables x and y of sort *Port* we have the formal rule

$$\text{PRE} \qquad x(y).v^* \xrightarrow{x(y)} v^*$$

where v^* is a formal variable of sort *Process*. The formal rule PRE expresses that process $x(y).p$ sends port name y via port x, and proceeds as process p. There is a subtle distinction between the two occurrences of y in PRE; in $x(y).v^*$ it is a binder of v^*, while in the label it is a free parameter. A notation of PRE in the vein of this article would be

$$send(x, y.v^*) \xrightarrow{(x,y)} v^*$$

From PRE we can deduce $x(y).t \xrightarrow{x(w)} t[w/y]$ for actual terms t of sort *Process* which do not contain any free occurrences of the actual variable w of sort *Port*, where $\_[w/y]$ is an explicit actual substitution. Namely, PRE yields $x(w).t[w/y] \xrightarrow{x(w)} t[w/y]$, and if w does not occur free in t, then $x(w).t[w/y]$ is α-convertible to $x(y).t$.

2.4 Proofs of Actual Rules

Examples 2.9 and 2.15 already showed that a TSS is used to prove that certain transitions hold. Now we give the precise definition of a proof from a TSS.

Definition 2.16 *A* proof *from a TSS T of an actual rule H/τ consists of an upwardly branching tree in which all upward paths are finite, where the nodes of the tree are labelled by positive and negative transitions such that:*

- *the root has label τ,*
- *if some node has label ℓ, and K is the set of labels of nodes directly above this node, then*
    1. *either $K = \emptyset$, and $\ell \in H$,*
    2. *or K/ℓ is a formal substitution instance of a formal rule in T.*

Example 2.17 In Example 2.9 we saw that the transition $\mu x.ax \xrightarrow{a} \mu x.ax$ can be proved from the TSS containing the formal rule for prefixing from CCS and the μ-rule. This proof is depicted in Figure 1.

Remark 2.18 Provability of an actual rule may depend in an essential way on the fact that terms are considered modulo α-conversion. For example, this was the case in Example 2.15, where the proof of the transition $x(y).t \xrightarrow{x(w)} t[w/y]$, with w not free in t, used α-conversion of $x(y).t$ to $x(w).t[w/y]$.

3 A Conservative Extension Theorem

In this section we present the theorem concerning conservative extensions. First, we define in a precise way what is a conservative extension. Then a string of technical definitions leads to the formulation and proof of the main theorem.

$$\begin{tikzcd}
 & \emptyset \arrow[d] & \\
\text{instance of prefixing rule} & a.\mu x.ax \xrightarrow{a} \mu x.ax \arrow[d] & \text{instance of } \mu\text{-rule} \\
 & \mu x.ax \xrightarrow{a} \mu x.ax &
\end{tikzcd}$$

Figure 1: A proof for $\mu x.ax \xrightarrow{a} \mu x.ax$

3.1 Well-Defined Sum

In order to be able to combine two TSSs, the function symbols and variables in the intersection of their signatures must have the same functionality in both signatures. Furthermore, if a relation or predicate symbol occurs in the two TSSs, then it must be either a relation or a predicate symbol in both TSSs. Therefore, we introduce the notion of a well-defined sum of two TSSs.

Definition 3.1 *Let T_0 and T_1 be TSSs over $(\Sigma_0, \mathcal{D}_0)$ and $(\Sigma_1, \mathcal{D}_1)$ respectively. Their sum (or union) $T_0 \oplus T_1$ is* well-defined *if*

- *each function symbol and each variable in $\Sigma_0 \cap \Sigma_1$ has the same functionality in both signatures;*
- *each element in $\mathcal{D}_0 \cap \mathcal{D}_1$ is either a relation or a predicate in both collections.*

In the remainder of this section we assume two TSSs T_0 and T_1 over $(\Sigma_0, \mathcal{D}_0)$ and $(\Sigma_1, \mathcal{D}_1)$ respectively, where $T_0 \oplus T_1$ is well-defined.

3.2 Conservative Extension

In the presence of negative hypotheses it is not straightforward to give meaning to a TSS. Several semantic notions have been introduced in the literature, such as two-valued and three-valued stable models, completeness, stratifications, and well-foundedness; see [27, 28] for an overview and a comparison of a wide range of such notions. Instead of restricting to one particular semantics, we define a stronger notion of a conservative extension, which can be regarded as a front-end to conservative extensions with respect to these semantic notions; see Section 3.8.

A conservative extension requires that an original TSS and its extension prove exactly the same actual rules N/τ with N a collection of negative transitions and the left-hand side of τ an original actual term.

Definition 3.2 *$T_0 \oplus T_1$ is an* (operationally) conservative extension *of T_0 if for each actual rule N/τ with*

- *N contains only negative transitions;*
- *the left-hand side of τ is in $\mathcal{T}(\Sigma_0)$;*
- *$T_0 \oplus T_1$ proves N/τ;*

we have that T_0 proves N/τ.

The notion of an operationally conservative extension of a TSS is related to an equivalence notion for TSSs that is used in [28, 20]: two TSSs are equivalent if they prove exactly the same actual rules N/τ where N contains only negative transitions.

We define a syntactic format for TSSs which ensures that a TSS $T_0 \oplus T_1$ is a conservative extension of T_0. But first we need to present several auxiliary definitions.

3.3 Fresh Formal Terms and Fresh Relations

A formal term in $\mathbb{F}(\Sigma_1)$ is called *fresh* if it incorporates a function symbol from $\Sigma_1 \backslash \Sigma_0$ outside its substitution harnesses.

Definition 3.3 *The formal terms in $\mathbb{F}(\Sigma_1)$ that are* fresh *are defined inductively as follows:*

- *$f(\vec{x}_1.t_1^*, ..., \vec{x}_n.t_n^*)$ is fresh if $f \in \Sigma_1 \backslash \Sigma_0$, or if some t_i^* is fresh;*
- *$t^*[s^*/x]$ is fresh if t^* is fresh.*

Example 3.4 *Let $\Sigma_0 = \{f\}$ and $\Sigma_1 = \{a, f\}$, where a is a constant and f is of arity one. Then $f(x.a[z^*/y])$ is fresh, but $f(x.z^*[a/y])$ is not fresh.*

Lemma 3.5 *$t^* \in \mathbb{F}(\Sigma_1)$ is fresh $\Rightarrow \sigma^*(t^*) \notin \mathbb{T}(\Sigma_0)$.*

Proof. By induction with respect to the size of t^*.

Definition 3.6 *Relations and predicates are called* fresh *if they are in $\mathcal{D}_1 \backslash \mathcal{D}_0$.*

3.4 The Collections $FV(t^*)$ and $EV(t^*)$

$FV(t^*)$ denotes the collection of formal variables that occur in the formal term t^*.

Definition 3.7 *The collections $FV(t^*)$ are defined inductively as follows.*

$$FV(x^*) = x^*,$$

$$FV(f(\vec{x}_1.t_1^*, ..., \vec{x}_n.t_n^*)) = FV(t_1^*) \cup ... \cup FV(t_n^*),$$

$$FV(t^*[s^*/x]) = FV(t^*) \cup FV(s^*).$$

Example 3.8 $FV(f(v.x^*[y^*/w])) = \{x^*, y^*\}$.

Lemma 3.9 *For formal terms $t^* \in \mathbb{F}(\Sigma_0)$ we have*

$$\sigma^*(x^*) \in \mathbb{T}(\Sigma_0) \text{ for all } x^* \in FV(t^*) \;\Rightarrow\; \sigma^*(t^*) \in \mathbb{T}(\Sigma_0).$$

Proof. By induction with respect to the size of t^*.

The converse of Lemma 3.9 does not hold. Namely, if $\sigma^*(t^*) \in \mathbb{T}(\Sigma_0)$, then it is possible for formal variables y^* that occur inside a substitution harness in t^* that $\sigma^*(y^*) \notin \mathbb{T}(\Sigma_0)$. This is illustrated by the following example.

Example 3.10 Let $\Sigma_0 = \{a\}$ and $\Sigma_1 = \{b\}$, where a and b are constants, and let $\sigma^*(x^*) = b$. Then $\sigma^*(a[x^*/y]) = a \in \mathbb{T}(\Sigma_0)$, but $\sigma^*(x^*) = b \notin \mathbb{T}(\Sigma_0)$.

In order to obtain a result converse to Lemma 3.9, we define a second, more restrictive collection $EV(t^*)$ of formal variables in a formal term t^*, which does not take into account formal variables that occur inside a substitution harness.

Definition 3.11 *The collections $EV(t^*)$ are defined inductively as follows.*

$$EV(x^*) = x^*,$$

$$EV(f(\vec{x}_1.t_1^*, ..., \vec{x}_n.t_n^*)) = EV(t_1^*) \cup ... \cup EV(t_n^*),$$

$$EV(t^*[s^*/x]) = EV(t^*).$$

Example 3.12 $EV(f(v.x^*[y^*/w])) = \{x^*\}$.

The definition of $EV(t^*)$ is motivated by the following lemma, which is the converse of Lemma 3.9, with FV replaced by EV.

Lemma 3.13 $\sigma^*(t^*) \in \mathbb{T}(\Sigma_0) \;\Rightarrow\; \sigma^*(x^*) \in \mathbb{T}(\Sigma_0)$ *for all* $x^* \in EV(t^*)$.

Proof. By induction with respect to the size of t^*.

3.5 Source-Dependency

Definition 3.14 *The formal term at the left-hand side of the conclusion of a formal rule is called the* source *of the formal rule.*

In this section we introduce the notion of source-dependency, modulo a set of sorts, for the formal variables in a formal rule. Source-dependency is an important ingredient of the conservativity theorem. In order to conclude that an extended TSS is conservative over an original TSS, we need to know that the formal variables in the original formal rules are source-dependent, modulo sorts for which there are no fresh terms. In practical cases, this criterion is sometimes neglected. For example, Nicollin and Sifakis [43] consider an extended TSS in which each formal rule in the extension contains a fresh operator in its source, and from this fact alone they conclude that it is a conservative extension. In general, however, this characteristic is not sufficient, as is shown in the next example.

Example 3.15 Let $\Sigma_0 = \{a\}$ and $\Sigma_1 = \{b\}$, where a and b are constants, and let R be a predicate. Consider the TSS over Σ_0 that consists of the formal rule x^*R/aR. Extend this TSS with the formal rule bR, which contains the fresh constant b in its source. Then aR holds in the extended TSS, but not in the original one, so this extension is not conservative.

Consider a formal rule r^*, which contains a formal variable x^*. Each of the following two properties ensures that for formal substitutions σ^* with the source of $\sigma^*(r^*)$ an original term, $\sigma^*(x^*)$ is also an original term.

- x^* occurs in the source of r^*, outside the substitution harnesses.
- There do not exist fresh terms of the same sort as x^*.

These two properties are captured in the first two cases of the definition of source-dependency modulo a set of sorts $\mathcal{S}$, respectively, where intuitively $\mathcal{S}$ consists of the sorts for which there do not exist fresh terms.

Definition 3.16 *For a formal rule r^*, and a collection $\mathcal{S}$ of sorts, the* source-dependent *formal variables* modulo $\mathcal{S}$ *in r^* are defined inductively as follows.*

1. *If t^* is the source of r^*, then all formal variables in $EV(t^*)$ are source-dependent in r^* modulo $\mathcal{S}$.*
2. *If $x^* \in FV(r^*)$ is of sort S for some $S \in \mathcal{S}$, then x^* is source-dependent in r^* modulo $\mathcal{S}$.*
3. *If $t_0^*R(t_1^*, ..., t_{n-1}^*)t_n^*$ is a hypothesis of r^*, and all formal variables in $FV(t_0^*)$ are source-dependent in r^* modulo $\mathcal{S}$, then all formal variables in $EV(t_i^*)$ for $i = 1, ..., n$ are source-dependent in r^* modulo $\mathcal{S}$.*
4. *If $t_0^*R(t_1^*, ..., t_{n-1}^*)$ is a hypothesis of r^*, and all formal variables in $FV(t_0^*)$ are source-dependent in r^* modulo $\mathcal{S}$, then all formal variables in $EV(t_i^*)$ for $i = 1, ..., n-1$ are source-dependent in r^* modulo $\mathcal{S}$.*

A formal variable is called source-dependent *if it is source-dependent modulo $\emptyset$.*

Source-dependency is a more liberal formulation of the syntactic criterion 'pure and well-founded' for formal variables in formal rules from Groote and Vaandrager [32]. In the setting without variable bindings, the notion of source-dependency was discovered independently by Van Glabbeek [26].

Example 3.17 We display the μ-rule, which was introduced in Example 2.9.

$$\frac{y^*[\mu x.y^*/x] \xrightarrow{a} z^*}{\mu x.y^* \xrightarrow{a} z^*}$$

Since the source of the μ-rule is $\mu x.y^*$ and $EV(\mu x.y^*) = \{y^*\}$, it follows that y^* is source-dependent (Definition 3.16 (1)). Since the μ-rule has a hypothesis $y^*[\mu x.y^*/x] \xrightarrow{a} z^*$, and

$$\begin{array}{l} FV(y^*[\mu x.y^*/x]) \;=\; FV(y^*) \cup FV(\mu x.y^*) \;=\; \{y^*\} \\ EV(z^*) \;=\; \{z^*\} \end{array}$$

it follows that z^* is also source-dependent (Definition 3.16 (3)).

3.6 The Formal Rule $\rho(r^*)$

Definition 3.18 *For each formal rule r^* in $T_0 \oplus T_1$, $\rho(r^*)$ denotes the formal rule that consists of the conclusion of r^*, together with those hypotheses of r^* for which the term at the left-hand side is in $\mathbb{F}(\Sigma_0)$.*

Example 3.19 Let $\Sigma_0 = \{a\}$ and $\Sigma_1 = \{b\}$, where a and b are constants, and let $\downarrow$ and $\uparrow$ be predicates. If r^* is the formal rule

$$\frac{a\!\downarrow \quad b\!\downarrow}{b\!\uparrow}$$

then $\rho(r^*)$ is $a\!\downarrow /b\!\uparrow$.

Note that if $r^* \in T_0$, then $\rho(r^*) = r^*$, simply because in this case all terms in r^* are in $\mathbb{F}(\Sigma_0)$.

3.7 The Main Theorem

Recall that we assume two TSSs T_0 and T_1 over $(\Sigma_0, \mathcal{D}_0)$ and $(\Sigma_1, \mathcal{D}_1)$ respectively, where $T_0 \oplus T_1$ is well-defined. Theorem 3.20 formulates sufficient criteria for $T_0 \oplus T_1$ to be a conservative extension of T_0.

Theorem 3.20 *Under the following conditions, $T_0 \oplus T_1$ is a conservative extension of T_0.*

1. *$\mathcal{S}$ is a collection of sorts such that for each $S \in \mathcal{S}$ there are no fresh actual terms in $\mathcal{T}(\Sigma_0 \oplus \Sigma_1)$ of sort S.*
2. *For each $r^* \in T_0$, all $x^* \in FV(r^*)$ are source-dependent in r^* modulo $\mathcal{S}$.*
3. *For each $r^* \in T_1$,*
   - *either the source of r^* is fresh,*
   - *or r^* has a hypothesis of the form $t_0^* R(t_1^*, ..., t_{n-1}^*) t_n^*$ or $t_0^* R(t_1^*, ..., t_{n-1}^*)$, where*
     - *$t_0^* \in \mathbb{F}(\Sigma_0)$;*
     - *all formal variables in $FV(t_0^*)$ are source-dependent in $\rho(r^*)$ modulo $\mathcal{S}$;*
     - *R or one of the terms $t_1^*, ..., t_n^*$ is fresh.*

In the proof of the conservativity theorem above we apply induction with respect to the *source distance* of a source-dependent formal variable x^* in a formal rule r^* modulo $\mathcal{S}$, being the minimal number of steps it takes to deduce that x^* is source-dependent in r^* modulo $\mathcal{S}$.

Definition 3.21 *Assume a formal rule r^* and a collection of sorts $\mathcal{S}$. For a formal variable $x^* \in FV(r^*)$ that is source-dependent modulo $\mathcal{S}$, its* source distance *$sd(r^*, \mathcal{S}, x^*)$ in r^* is defined as follows.*

- *If t^* is the source of r^* and $x^* \in EV(t^*)$, then $sd(r^*, \mathcal{S}, x^*) \leq n$ holds for all naturals n.*

- *If x^* is of sort S for some $S \in \mathcal{S}$, then $sd(r^*, \mathcal{S}, x^*) \leq n$ holds for all naturals n.*

- *If $t_0^* R(t_1^*, ..., t_{n-1}^*) t_n^*$ is a hypothesis of r^*, and $sd(r^*, \mathcal{S}, x^*) \leq n$ holds for all $x^* \in FV(t_0^*)$, then $sd(r^*, \mathcal{S}, y^*) \leq n+1$ holds for all $y^* \in EV(t_1^*) \cup ... \cup EV(t_n^*)$.*

- *If $t_0^* R(t_1^*, ..., t_{n-1}^*)$ is a hypothesis of r^*, and $sd(r^*, \mathcal{S}, x^*) \leq n$ holds for all $x^* \in FV(t_0^*)$, then $sd(r^*, \mathcal{S}, y^*) \leq n+1$ holds for all $y^* \in EV(t_1^*) \cup ... \cup EV(t_{n-1}^*)$.*

Finally, $sd(r^*, \mathcal{S}, x^*) = n$ if n is the smallest number such that $sd(r^*, \mathcal{S}, x^*) \leq n$.

Proof of Theorem 3.20. Suppose that there exists a proof P from $T_0 \oplus T_1$ for an actual rule $N/t_0 R(t_1, ..., t_{n-1}) t_n$, where N consists of negative transitions and $t_0 \in \mathcal{T}(\Sigma_0)$. We need to prove that P is a proof from T_0, which we do by ordinal induction A on the length of P. (The case that $T_0 \oplus T_1$ proves an actual rule $N/t_0 R(t_1, ..., t_{n-1})$, where N consists of negative transitions and $t_0 \in \mathcal{T}(\Sigma_0)$, can be dealt with in a similar fashion.)

Let P have length α, and suppose that we have already proved the case for ordinals smaller than α. The last step in P is constituted by a formal rule $r^* \in T_0 \oplus T_1$ with a conclusion of the form $p_0^* R(p_1^*, ..., p_{n-1}^*) p_n^*$ together with a formal substitution $\sigma^* : \mathcal{V}^* \to \mathbb{T}(\Sigma_0 \oplus \Sigma_1)$, where $\sigma^*(p_0^*) = t_0$.

First, we show that $\sigma^*(x^*) \in \mathbb{T}(\Sigma_0)$ for all x^* that are source-dependent in $\rho(r^*)$ modulo $\mathcal{S}$, by induction B on the source distance of x^* in $\rho(r^*)$ (see Definition 3.21).

1. $sd(\rho(r^*), \mathcal{S}, x^*) = 0$.

   This means that either $x^* \in EV(p_0^*)$, or x^* is of sort S for some $S \in \mathcal{S}$.

   Suppose that $x^* \in EV(p_0^*)$. Since $\sigma^*(p_0^*) = t_0$ is in $\mathcal{T}(\Sigma_0)$, Lemma 3.13 then yields $\sigma^*(x^*) \in \mathbb{T}(\Sigma_0)$.

   Suppose that x^* is of sort $S \in \mathcal{S}$. By Assumption 1 of Theorem 3.20 there are no fresh actual terms of sort S, so in this case also $\sigma^*(x^*) \in \mathbb{T}(\Sigma_0)$.

2. $sd(\rho(r^*), \mathcal{S}, x^*) = k+1$.

   By definition there is a hypothesis $q_0^* U(q_1^*, ..., q_{m-1}^*) q_m^*$ or $q_0^* U(q_1^*, ..., q_{m-1}^*)$ of $\rho(r^*)$ such that $x^* \in EV(q_i^*)$ for some $i = 1, ..., m$ and $sd(\rho(r^*), \mathcal{S}, y^*) \leq k$ for all $y^* \in FV(q_0^*)$. Induction B implies that $\sigma^*(y^*) \in \mathbb{T}(\Sigma_0)$ for all $y^* \in FV(q_0^*)$. Furthermore, Definition 3.18 of $\rho(r^*)$ ensures that $q_0^* \in \mathbb{F}(\Sigma_0)$, so Lemma 3.9 yields $\sigma^*(q_0^*) \in \mathcal{T}(\Sigma_0)$. The transition $\sigma^*(q_0^* U(q_1^*, ..., q_{m-1}^*) q_m^*)$ or $\sigma^*(q_0^* U(q_1^*, ..., q_{m-1}^*))$ is proved by a strict sub-proof of P, so then ordinal induction A implies that T_0 proves this transition. In particular, $\sigma^*(q_i^*) \in \mathcal{T}(\Sigma_0)$ for $i = 1, ..., m$. Since $x^* \in EV(q_i^*)$ for some $i = 1, ..., m$, Lemma 3.13 yields $\sigma^*(x^*) \in \mathbb{T}(\Sigma_0)$.

Next, we show that r^* is in T_0. Suppose not, so let $r^* \in T_1$; we deduce a contradiction. Since $\sigma^*(p_0^*) = t_0$ is in $\mathcal{T}(\Sigma_0)$, Lemma 3.5 implies that p_0^* is not fresh. Then by Assumption 3 of Theorem 3.20 there is a hypothesis in r^* of the form $q_0^* U(q_1^*, ..., q_{m-1}^*) q_m^*$ or $q_0^* U(q_1^*, ..., q_{m-1}^*)$, where either U or some q_i^* for $i = 1, ..., m$ is fresh, and $q_0^* \in \mathbb{F}(\Sigma_0)$, and all formal variables in $FV(q_0^*)$ are source-dependent in $\rho(r^*)$ modulo $\mathcal{S}$.

If q_i^* is fresh for some $i = 1, ..., m$, then Lemma 3.5 says that $\sigma^*(q_i^*) \notin \mathcal{T}(\Sigma_0)$. Hence, since either U is fresh or $\sigma^*(q_i^*) \notin \mathcal{T}(\Sigma_0)$ for some $i = 1, ..., m$, the sub-proof of P of $N/\sigma^*(q_0^* U(q_1^*, ..., q_{m-1}^*) q_m^*)$ or $N/\sigma^*(q_0^* U(q_1^*, ..., q_{m-1}^*))$ cannot be a proof from T_0. So according to ordinal induction A, $\sigma^*(q_0^*) \notin \mathcal{T}(\Sigma_0)$. Since $q_0^* \in \mathbb{F}(\Sigma_0)$, Lemma 3.9 yields $\sigma^*(x^*) \notin \mathbb{T}(\Sigma_0)$ for some $x^* \in FV(q_0^*)$. Then x^* is not source-dependent in $\rho(r^*)$ modulo $\mathcal{S}$. Contradiction.

So apparently r^* is in T_0. Then $\rho(r^*) = r^*$ (see Section 3.6), so $\sigma^*(x^*) \in \mathbb{T}(\Sigma_0)$ for all x^* that are source-dependent in r^* modulo $\mathcal{S}$. According to Assumption 2 of Theorem 3.20 all variables in $FV(r^*)$ are source-dependent in r^* modulo $\mathcal{S}$. Thus, $\sigma^*(r^*)$ contains only closed actual terms from $\mathcal{T}(\Sigma_0)$. In particular, for each positive hypothesis h^* in r^*, the left-hand side of $\sigma^*(h^*)$ is in $\mathcal{T}(\Sigma_0)$. Then induction A says that the sub-proof of P for $N/\sigma^*(h^*)$ is a proof from T_0. Since the last step (with r^* and σ^*) is in T_0 too, P is a proof from T_0. □

3.8 Three-Valued Stable Models

We use *three-valued stable models*, introduced by Przymusinski [45], to give a semantics to TSSs with negative hypotheses, and discuss how the conservative extension property as formulated in Definition 3.2 implies a conservativity result for these models.

Definition 3.22 *A collection of negative transitions N* holds *for a set of positive transitions $\mathtt{P}$, denoted by $\mathtt{P} \models N$, if for each $t_0 \neg R(t_1, ..., t_{n-1}) \in N$ we have*

- *either $t_0 R(t_1, ..., t_{n-1}) t \notin \mathtt{P}$ for all actual terms t if R is a relation;*
- *or $t_0 R(t_1, ..., t_{n-1}) \notin \mathtt{P}$ if R is a predicate.*

A three-valued stable model partitions the collection of positive transitions into three disjoint sets: the set $\mathtt{C}$ of transitions that are *certainly* true, the set $\mathtt{U}$ of transitions for which it is *unknown* whether or not they are true, and the set of remaining transitions that are false. Such a partitioning (which is determined by $\langle \mathtt{C}, \mathtt{U} \rangle$) constitutes a three-valued stable model for TSS T if:

- a positive transition τ is in $\mathtt{C}$ if and only if T proves an actual rule N/τ where N contains only negative transitions and $\mathtt{C} \cup \mathtt{U} \models N$;
- a positive transition τ is in $\mathtt{C} \cup \mathtt{U}$ if and only if T proves an actual rule N/τ where N contains only negative transitions and $\mathtt{C} \models N$.

A TSS may allow more than one three-valued stable model.

Example 3.23 Assume two constants a and b, and a predicate R. The TSS that consists of the formal rules $b \neg R / aR$ and $a \neg R / bR$ allows three three-valued stable models, namely $\langle \emptyset, \{aR, bR\} \rangle$ and $\langle \{aR\}, \emptyset \rangle$ and $\langle \{bR\}, \emptyset \rangle$.

The conservative extension notion as formulated in Definition 3.2 implies a conservativity property for three-valued stable models. Namely, if an extended TSS is conservative over the original TSS, in the sense of Definition 3.2, and if a three-valued stable model for the extended TSS is restricted to those positive transitions that have an original term as left-hand side, then the result is a three-valued stable model for the original TSS.

Theorem 3.24 *Let $T_0 \oplus T_1$ be a conservative extension of T_0. If $\langle \mathtt{C}, \mathtt{U} \rangle$ is a three-valued stable model for $T_0 \oplus T_1$, then*

$$\begin{array}{rcl} \mathtt{C}' & = & \{\tau \in \mathtt{C} \,|\, \textit{the left-hand side of } \tau \textit{ is in } \mathcal{T}(\Sigma_0)\} \\ \mathtt{U}' & = & \{\tau \in \mathtt{U} \,|\, \textit{the left-hand side of } \tau \textit{ is in } \mathcal{T}(\Sigma_0)\} \end{array}$$

is a three-valued stable model for T_0.

Proof. We have to check that

1. $\tau \in \mathtt{C}'$ if and only if T_0 proves an actual rule N/τ with $\mathtt{C}' \cup \mathtt{U}' \models N$;
2. $\tau \in \mathtt{C}' \cup \mathtt{U}'$ if and only if T_0 proves an actual rule N/τ with $\mathtt{C}' \models N$.

The proofs of these two statements are almost identical. We spell out both proofs, in order to exhibit their subtle distinctions.

1a Assume that there is a proof from T_0 for an actual rule N/τ, where N contains only negative transitions, and $\mathtt{C}' \cup \mathtt{U}' \models N$. We show that $\tau \in \mathtt{C}'$.

   Since T_0 proves N/τ, clearly N and τ involve only closed actual terms from $\mathcal{T}(\Sigma_0)$. Furthermore, the proof for N/τ from T_0 is also a proof from $T_0 \oplus T_1$.

   Consider a negative transition $t_0 \neg R(t_1, ..., t_{n-1})$ in N. Since $\mathtt{C}' \cup \mathtt{U}' \models N$, either $t_0 R(t_1, ..., t_{n-1}) t \notin \mathtt{C}' \cup \mathtt{U}'$ for all closed actual terms $t \in \mathcal{T}(\Sigma_0 \oplus \Sigma_1)$ (if R is a relation), or $t_0 R(t_1, ..., t_{n-1}) \notin \mathtt{C}' \cup \mathtt{U}'$ (if R is a predicate). Since N involves only closed actual terms from $\mathcal{T}(\Sigma_0)$, in particular $t_0 \in \mathcal{T}(\Sigma_0)$. Thus, by definition of $\mathtt{C}'$ and $\mathtt{U}'$, either $t_0 R(t_1, ..., t_{n-1}) t \notin \mathtt{C} \cup \mathtt{U}$ for all $t \in \mathcal{T}(\Sigma_0 \oplus \Sigma_1)$, or $t_0 R(t_1, ..., t_{n-1}) \notin \mathtt{C} \cup \mathtt{U}$, respectively. Hence $\mathtt{C} \cup \mathtt{U} \models N$. Since $\langle \mathtt{C}, \mathtt{U} \rangle$ constitutes a three-valued stable model for $T_0 \oplus T_1$, and there is a proof from $T_0 \oplus T_1$ for N/τ, this implies $\tau \in \mathtt{C}$.

   Since τ contains only actual terms from $\mathcal{T}(\Sigma_0)$, in particular its left-hand side is in $\mathcal{T}(\Sigma_0)$, and so $\tau \in \mathtt{C}'$.

1b Assume that $\tau \in \mathtt{C}'$. We show that there is a proof from T_0 for an actual rule N/τ, where N contains only negative transitions, and $\mathtt{C}' \cup \mathtt{U}' \models N$.

   $\tau \in \mathtt{C}' \subseteq \mathtt{C}$, and $\langle \mathtt{C}, \mathtt{U} \rangle$ constitutes a three-valued stable model for $T_0 \oplus T_1$. So there exists a proof from $T_0 \oplus T_1$ for an actual rule N/τ, where N consists of negative transitions, and $\mathtt{C} \cup \mathtt{U} \models N$. Since $T_0 \oplus T_1$ is a conservative extension of T_0, and the left-hand side of τ is in $\mathcal{T}(\Sigma_0)$, there exists a proof for N/τ from T_0. Finally, $\mathtt{C}' \cup \mathtt{U}' \subseteq \mathtt{C} \cup \mathtt{U}$ and $\mathtt{C} \cup \mathtt{U} \models N$ together imply $\mathtt{C}' \cup \mathtt{U}' \models N$.

2a Assume that there is a proof from T_0 for an actual rule N/τ, where N contains only negative transitions, and $\mathtt{C}' \models N$. We show that $\tau \in \mathtt{C}' \cup \mathtt{U}'$.

   Since T_0 proves N/τ, clearly N and τ involve only closed actual terms from $\mathcal{T}(\Sigma_0)$. Furthermore, the proof for N/τ from T_0 is also a proof from $T_0 \oplus T_1$.

Consider a negative transition $t_0 \neg R(t_1, ..., t_{n-1})$ in N. Since $\mathtt{C}' \models N$, either $t_0 R(t_1, ..., t_{n-1}) t \notin \mathtt{C}'$ for all closed actual terms $t \in \mathcal{T}(\Sigma_0 \oplus \Sigma_1)$ (if R is a relation), or $t_0 R(t_1, ..., t_{n-1}) \notin \mathtt{C}'$ (if R is a predicate). Since N involves only closed actual terms from $\mathcal{T}(\Sigma_0)$, in particular $t_0 \in \mathcal{T}(\Sigma_0)$. Thus, by definition of $\mathtt{C}'$, either $t_0 R(t_1, ..., t_{n-1}) t \notin \mathtt{C}$ for all $t \in \mathcal{T}(\Sigma_0 \oplus \Sigma_1)$, or $t_0 R(t_1, ..., t_{n-1}) \notin \mathtt{C}$, respectively. Hence $\mathtt{C} \models N$. Since $\langle \mathtt{C}, \mathtt{U} \rangle$ constitutes a three-valued stable model for $T_0 \oplus T_1$, and there is a proof from $T_0 \oplus T_1$ for N/τ, this implies $\tau \in \mathtt{C} \cup \mathtt{U}$.

Since τ contains only actual terms from $\mathcal{T}(\Sigma_0)$, in particular its left-hand side is in $\mathcal{T}(\Sigma_0)$, and so $\tau \in \mathtt{C}' \cup \mathtt{U}'$.

2b Assume that $\tau \in \mathtt{C}' \cup \mathtt{U}'$. We show that there is a proof from T_0 for an actual rule N/τ, where N contains only negative transitions, and $\mathtt{C}' \models N$.

$\tau \in \mathtt{C}' \cup \mathtt{U}' \subseteq \mathtt{C} \cup \mathtt{U}$, and $\langle \mathtt{C}, \mathtt{U} \rangle$ constitutes a three-valued stable model for $T_0 \oplus T_1$. So there exists a proof from $T_0 \oplus T_1$ for an actual rule N/τ, where N consists of negative transitions, and $\mathtt{C} \models N$. Since $T_0 \oplus T_1$ is a conservative extension of T_0, and the left-hand side of τ is in $\mathcal{T}(\Sigma_0)$, there exists a proof for N/τ from T_0. Finally, $\mathtt{C}' \subseteq \mathtt{C}$ and $\mathtt{C} \models N$ together imply $\mathtt{C}' \models N$. $\square$

The reverse of Theorem 3.24 also holds, in the following sense. If an extended TSS is conservative over the original TSS, then each three-valued stable model for the original TSS can be obtained by restricting some three-valued stable model for the extended TSS to those positive transitions that have an original term as left-hand side.

Theorem 3.25 *Let $T_0 \oplus T_1$ be a conservative extension of T_0. If $\langle \mathtt{C}, \mathtt{U} \rangle$ is a three-valued stable model for T_0, then there exists a three-valued stable model $\langle \mathtt{C}', \mathtt{U}' \rangle$ for $T_0 \oplus T_1$ such that*

$$\begin{array}{rcl} \mathtt{C} & = & \{\tau \in \mathtt{C}' \mid \textit{the left-hand side of } \tau \textit{ is in } \mathcal{T}(\Sigma_0)\} \\ \mathtt{U} & = & \{\tau \in \mathtt{U}' \mid \textit{the left-hand side of } \tau \textit{ is in } \mathcal{T}(\Sigma_0)\}. \end{array}$$

Proof. We construct pairs of disjoint sets of positive transitions $\langle \mathtt{C}_\alpha, \mathtt{U}_\alpha \rangle$ for ordinals α, using ordinal induction, and show that these pairs converge to a suitable three-valued stable model for $T_0 \oplus T_1$.

- $\mathtt{C}_0 = \mathtt{C}$, and $\mathtt{U}_0$ consists of $\mathtt{U}$ together with all positive transitions that do not have a term from $\mathcal{T}(\Sigma_0)$ as left-hand side.

- For ordinals α, $\langle \mathtt{C}_{\alpha+1}, \mathtt{U}_{\alpha+1} \rangle$ is constructed from $\langle \mathtt{C}_\alpha, \mathtt{U}_\alpha \rangle$ as follows. A positive transition τ is in $\mathtt{C}_{\alpha+1}$ iff $T_0 \oplus T_1$ proves an actual rule N/τ where N contains only negative transitions and $\mathtt{C}_\alpha \cup \mathtt{U}_\alpha \models N$. Furthermore, a positive transition τ is in $\mathtt{C}_{\alpha+1} \cup \mathtt{U}_{\alpha+1}$ iff $T_0 \oplus T_1$ proves an actual rule N/τ where N contains only negative transitions and $\mathtt{C}_\alpha \models N$.

- For limit ordinals λ we define $\mathtt{C}_\lambda = \cup_{\alpha<\lambda} \mathtt{C}_\alpha$ and $\mathtt{U}_\lambda = \cap_{\alpha<\lambda} \mathtt{U}_\alpha$.

First, we prove for all ordinals α:

I. $\mathtt{C} = \{\tau \in \mathtt{C}_\alpha \mid$ the left-hand side of τ is in $\mathcal{T}(\Sigma_0)\}$

II. $\mathtt{U} = \{\tau \in \mathtt{U}_\alpha \mid$ the left-hand side of τ is in $\mathcal{T}(\Sigma_0)\}$

Proof. We prove both equalities in parallel, using ordinal induction with respect to α. The case $\alpha = 0$ follows immediately from the definitions of $\mathtt{C}_0$ and $\mathtt{U}_0$. We focus on the inductive case.

Ia Let $\tau \in \mathtt{C}$. Then the left-hand side of τ is in $\mathcal{T}(\Sigma_0)$. We show that $\tau \in \mathtt{C}_\alpha$.

If α is a limit ordinal, then by induction $\tau \in \mathtt{C}_\beta$ for $\beta < \alpha$, so $\tau \in \mathtt{C}_\alpha$.

Let α be a non-limit ordinal. $\tau \in \mathtt{C}$ implies that T_0 proves an actual rule N/τ where N contains only negative transitions and $\mathtt{C}\cup\mathtt{U} \models N$. Then $T_0 \oplus T_1$ also proves N/τ. Furthermore, the left-hand sides of transitions in N are all in $\mathcal{T}(\Sigma_0)$, so by induction $\mathtt{C} \cup \mathtt{U} \models N$ implies $\mathtt{C}_{\alpha-1} \cup \mathtt{U}_{\alpha-1} \models N$. Hence, $\tau \in \mathtt{C}_\alpha$.

Ib Let $\tau \in \mathtt{C}_\alpha$ with its left-hand side in $\mathcal{T}(\Sigma_0)$. We show that $\tau \in \mathtt{C}$.

If α is a limit ordinal, then $\tau \in \mathtt{C}_\beta$ for some $\beta < \alpha$, so by induction $c \in \mathtt{C}$.

If α is not a limit ordinal, then $\tau \in \mathtt{C}_\alpha$ yields that $T_0 \oplus T_1$ proves an actual rule N/τ where N contains only negative transitions and $\mathtt{C}_{\alpha-1}\cup\mathtt{U}_{\alpha-1} \models N$. Since $T_0 \oplus T_1$ is a conservative extension of T_0 and the left-hand side of τ is in $\mathcal{T}(\Sigma_0)$, T_0 also proves N/τ. Furthermore, by induction $\mathtt{C} \cup \mathtt{U} \subseteq \mathtt{C}_{\alpha-1} \cup \mathtt{U}_{\alpha-1} \models N$. Hence, $\tau \in \mathtt{C}$.

IIa Let $\tau \in \mathtt{U}$. Then the left-hand side of τ is in $\mathcal{T}(\Sigma_0)$. We show that $\tau \in \mathtt{U}_\alpha$.

If α is a limit ordinal, then by induction $\tau \in \mathtt{U}_\beta$ for all $\beta < \alpha$, so $\tau \in \mathtt{U}_\alpha$.

Let α be a non-limit ordinal. $\tau \in \mathtt{U}$ implies that T_0 proves an actual rule N/τ where N contains only negative transitions and $\mathtt{C} \models N$ and $\mathtt{C}\cup\mathtt{U} \not\models N$. Then $T_0 \oplus T_1$ also proves N/τ. Furthermore, the left-hand sides of transitions in N are all in $\mathcal{T}(\Sigma_0)$, so by induction $\mathtt{C} \models N$ implies $\mathtt{C}_{\alpha-1} \models N$. Finally, by induction $\mathtt{C}_{\alpha-1} \cup \mathtt{U}_{\alpha-1} \supseteq \mathtt{C} \cup \mathtt{U} \not\models N$. Hence, $\tau \in \mathtt{U}_\alpha$.

IIb Let $\tau \in \mathtt{U}_\alpha$ with its left-hand side in $\mathcal{T}(\Sigma_0)$. We show that $\tau \in \mathtt{U}$.

If α is a limit ordinal, then $\tau \in \mathtt{U}_\beta$ for $\beta < \alpha$, so by induction $\tau \in \mathtt{U}$.

If α is not a limit ordinal, then $\tau \in \mathtt{U}_\alpha$ yields that $T_0 \oplus T_1$ proves an actual rule N/τ where N contains only negative transitions and $\mathtt{C}_{\alpha-1} \models N$ and $\mathtt{C}_{\alpha-1} \cup \mathtt{U}_{\alpha-1} \not\models N$. Since $T_0 \oplus T_1$ is a conservative extension of T_0 and the left-hand side of τ is in $\mathcal{T}(\Sigma_0)$, T_0 also proves N/τ. Furthermore, the left-hand sides of transitions in N are all in $\mathcal{T}(\Sigma_0)$, so by induction $\mathtt{C}_{\alpha-1} \cup \mathtt{U}_{\alpha-1} \not\models N$ implies $\mathtt{C} \cup \mathtt{U} \not\models N$. Finally, by induction $\mathtt{C} \subseteq \mathtt{C}_{\alpha-1} \models N$. Hence, $\tau \in \mathtt{U}$.

Next, we prove three inclusions for ordinals α and β with $\alpha < \beta$. The last two inclusions enable us to apply the well-known fixpoint theorem of Knaster-Tarski [51]. (The first inclusion is needed in the proof of the second inclusion.)

1. $\mathtt{C}_\alpha \cup \mathtt{U}_\alpha \supseteq \mathtt{C}_\beta \cup \mathtt{U}_\beta$;

2. $\mathtt{C}_\alpha \subseteq \mathtt{C}_\beta$;

3. $\mathtt{U}_\alpha \supseteq \mathtt{U}_\beta$.

Proof. First we prove inclusions (1) and (2) in parallel, using ordinal induction with respect to (α, β), where $(\alpha', \beta') < (\alpha, \beta)$ if either $\beta' < \beta$, or $\beta' = \beta$ and $\alpha' < \alpha$. We start with the base case where $\alpha = 0$.

- $\mathtt{C}_0 \cup \mathtt{U}_0 \supseteq \mathtt{C}_\beta \cup \mathtt{U}_\beta$.
  Let $\tau \in \mathtt{C}_\beta \cup \mathtt{U}_\beta$. If the left-hand side of τ is not in $\mathcal{T}(\Sigma_0)$, then $\tau \in \mathtt{U}_0 \subseteq \mathtt{C}_0 \cup \mathtt{U}_0$.
  If the left-hand side of τ is in $\mathcal{T}(\Sigma_0)$, then by equalities (I) and (II) $\tau \in \mathtt{C} \cup \mathtt{U} \subseteq \mathtt{C}_0 \cup \mathtt{U}_0$.
- $\mathtt{C}_0 \subseteq \mathtt{C}_\beta$.
  Since $\mathtt{C}_0 = \mathtt{C}$, this follows from equality (I).

Next, we prove inclusions (1) and (2) for the inductive case.

- $\mathtt{C}_\alpha \cup \mathtt{U}_\alpha \supseteq \mathtt{C}_\beta \cup \mathtt{U}_\beta$.
  Let $\tau \in \mathtt{C}_\beta \cup \mathtt{U}_\beta$; we show that $\tau \in \mathtt{C}_\alpha \cup \mathtt{U}_\alpha$. We distinguish several cases, depending on whether or not α and β are limit ordinals.
  Case 1: α is a limit ordinal.
  Inclusion (1) yields that $\tau \in \mathtt{C}_\beta \cup \mathtt{U}_\beta \subseteq \mathtt{C}_\gamma \cup \mathtt{U}_\gamma$ for all $\gamma < \alpha$. We distinguish two cases.
  Case 1.1: $\tau \in \mathtt{C}_\gamma$ for some $\gamma < \alpha$.
  Then inclusion (2) yields $\tau \in \mathtt{C}_\gamma \subseteq \mathtt{C}_\alpha$.
  Case 1.2: $\tau \in \mathtt{U}_\gamma$ for all $\gamma < \alpha$.
  Then $\tau \in \cap_{\gamma<\alpha}\mathtt{U}_\gamma = \mathtt{U}_\alpha$.
  Case 2: β is a limit ordinal. We distinguish two cases.
  Case 2.1: $\tau \in \mathtt{U}_\beta$.
  Then $\tau \in \mathtt{U}_\gamma$ for all $\gamma < \beta$, so in particular $\tau \in \mathtt{U}_\alpha$.
  Case 2.2: $\tau \in \mathtt{C}_\beta$.
  Then $\tau \in \mathtt{C}_\gamma$ for some $\gamma < \beta$. We distinguish two cases.
  Case 2.2.1: $\gamma \leq \alpha$.
  Then by inclusion (2) $\tau \in \mathtt{C}_\gamma \subseteq \mathtt{C}_\alpha$.
  Case 2.2.2: $\gamma \geq \alpha$.
  Then by inclusion (1) $\tau \in \mathtt{C}_\gamma \subseteq \mathtt{C}_\gamma \cup \mathtt{U}_\gamma \subseteq \mathtt{C}_\alpha \cup \mathtt{U}_\alpha$.
  Case 3: Both α and β are not limit ordinals.
  Since $\tau \in \mathtt{C}_\beta \cup \mathtt{U}_\beta$, $T_0 \oplus T_1$ proves an actual rule N/τ where N contains only negative transitions and $\mathtt{C}_{\beta-1} \models N$. Inclusion (2) yields $\mathtt{C}_{\alpha-1} \subseteq \mathtt{C}_{\beta-1} \models N$. Hence, $\tau \in \mathtt{C}_\alpha \cup \mathtt{U}_\alpha$.
- $\mathtt{C}_\alpha \subseteq \mathtt{C}_\beta$.
  Let $\tau \in \mathtt{C}_\alpha$; we show that $\tau \in \mathtt{C}_\beta$. We distinguish several cases, depending on whether or not α and β are limit ordinals.
  Case 1: α is a limit ordinal.
  Inclusion (2) yields $\mathtt{C}_\gamma \subseteq \mathtt{C}_\beta$ for all $\gamma < \alpha$, so $\mathtt{C}_\alpha = \cup_{\gamma<\alpha}\mathtt{C}_\gamma \subseteq \mathtt{C}_\beta$.
  Case 2: β is a limit ordinal.
  Then $\mathtt{C}_\alpha \subseteq \cup_{\gamma<\beta}\mathtt{C}_\gamma = \mathtt{C}_\beta$.
  Case 3: Both α and β are not limit ordinals. Since $\tau \in \mathtt{C}_\alpha$, $T_0 \oplus T_1$ proves an actual rule N/τ where N contains only negative transitions and $\mathtt{C}_{\alpha-1} \cup \mathtt{U}_{\alpha-1} \models N$. Inclusion (1) yields $\mathtt{C}_{\beta-1} \cup \mathtt{U}_{\beta-1} \subseteq \mathtt{C}_{\alpha-1} \cup \mathtt{U}_{\alpha-1} \models N$. Hence, $\tau \in \mathtt{C}_\beta$.

Finally, we prove inclusion (3).

- $\mathtt{C}_\alpha \cup \mathtt{U}_\alpha \supseteq \mathtt{C}_\beta \cup \mathtt{U}_\beta$ and $\mathtt{C}_\alpha \subseteq \mathtt{C}_\beta$ together yield

$$\mathtt{U}_\beta = (\mathtt{C}_\beta \cup \mathtt{U}_\beta)\backslash\mathtt{C}_\beta \subseteq (\mathtt{C}_\alpha \cup \mathtt{U}_\alpha)\backslash\mathtt{C}_\alpha = \mathtt{U}_\alpha.$$

Owing to inclusions (2) and (3), the Knaster-Tarski theorem yields that there exists an ordinal α such that $\mathtt{C}_\alpha = \mathtt{C}_{\alpha+1}$ and $\mathtt{U}_\alpha = \mathtt{U}_{\alpha+1}$. We show that $\langle \mathtt{C}_\alpha, \mathtt{U}_\alpha \rangle$ is a three-valued stable model for $T_0 \oplus T_1$.

- By definition of $\mathtt{C}_{\alpha+1}$, $\tau \in \mathtt{C}_\alpha (= \mathtt{C}_{\alpha+1})$ iff $T_0 \oplus T_1$ proves an actual rule N/τ where N contains only negative transitions and $\mathtt{C}_\alpha \cup \mathtt{U}_\alpha \models N$.

- By definition of $\mathtt{C}_{\alpha+1} \cup \mathtt{U}_{\alpha+1}$, $\tau \in \mathtt{C}_\alpha \cup \mathtt{U}_\alpha (= \mathtt{C}_{\alpha+1} \cup \mathtt{U}_{\alpha+1})$ iff $T_0 \oplus T_1$ proves an actual rule N/τ where N contains only negative transitions and $\mathtt{C}_\alpha \models N$.

Owing to equalities (I) and (II), $\langle \mathtt{C}_\alpha, \mathtt{U}_\alpha \rangle$ is the desired three-valued stable model for $T_0 \oplus T_1$. $\square$

Przymusinski [45] noted that each TSS allows a least three-valued stable model, in the sense that the set of unknown transitions is maximal. (The construction of this least three-valued stable model is similar to the limit construction in the proof of Theorem 3.25, with the distinction that $\mathtt{C}_0$ is taken to be empty and $\mathtt{U}_0$ is taken to be the set of all positive transitions.) Przymusinski proved that the least three-valued stable model coincides with the *well-founded* semantics of Van Gelder, Ross, and Schlipf [24].

Theorem 3.26 *Let $T_0 \oplus T_1$ be a conservative extension of T_0. If $\langle \mathtt{C}, \mathtt{U} \rangle$ is the least three-valued stable model for $T_0 \oplus T_1$, then*

$$\begin{array}{rcl} \mathtt{C}' & = & \{\tau \in \mathtt{C} \mid \textit{the left-hand side of } \tau \textit{ is in } \mathcal{T}(\Sigma_0)\} \\ \mathtt{U}' & = & \{\tau \in \mathtt{U} \mid \textit{the left-hand side of } \tau \textit{ is in } \mathcal{T}(\Sigma_0)\} \end{array}$$

is the least three-valued stable model for T_0.

Proof. According to Theorem 3.24, $\langle \mathtt{C}', \mathtt{U}' \rangle$ is a three-valued stable model for T_0. Consider an arbitrary three-valued stable model $\langle \overline{\mathtt{C}}', \overline{\mathtt{U}}' \rangle$ for T_0. According to Theorem 3.25 there exists a three-valued stable model $\langle \overline{\mathtt{C}}, \overline{\mathtt{U}} \rangle$ for $T_0 \oplus T_1$ such that

$$\begin{array}{rcl} \overline{\mathtt{C}}' & = & \{\tau \in \overline{\mathtt{C}} \mid \text{the left-hand side of } \tau \text{ is in } \mathcal{T}(\Sigma_0)\} \\ \overline{\mathtt{U}}' & = & \{\tau \in \overline{\mathtt{U}} \mid \text{the left-hand side of } \tau \text{ is in } \mathcal{T}(\Sigma_0)\}. \end{array}$$

Since $\langle \mathtt{C}, \mathtt{U} \rangle$ is the least three-valued stable model for $T_0 \oplus T_1$ we have $\overline{\mathtt{U}} \subseteq \mathtt{U}$, and so $\overline{\mathtt{U}}' \subseteq \mathtt{U}'$. Hence, $\langle \mathtt{C}', \mathtt{U}' \rangle$ is the least three-valued stable model for T_0. $\Box$

The notion of a *(two-valued) stable model* stems from Gelfond and Lifschitz [25] in the setting of logic programming, and was adapted to structured operational semantics by Bol and Groote [14]. A two-valued stable model is a three-valued stable model of the form $\langle \mathtt{C}, \emptyset \rangle$. It is easy to see that Theorem 3.24 also holds for two-valued instead of three-valued stable models. The following example, however, shows that Theorem 3.25 does *not* hold for two-valued stable models.

Example 3.27 *Let T_0 be the empty TSS. T_0 allows the two-valued stable model $\langle \emptyset, \emptyset \rangle$.*
Let a be a constant and R a predicate, and let T_1 consist of the single rule $a\neg R/aR$. According to Theorem 3.20, $T_0 \oplus T_1$ is a conservative extension of T_0. However, $T_0 \oplus T_1$ does not allow a two-valued stable model, but only the three-valued stable model $\langle \emptyset, \{aR\} \rangle$.

Van Glabbeek [28] argued that a good way to give meaning to TSSs with negative hypotheses is through the notion of *completeness.* A TSS is complete if its least three-valued stable model is a two-valued stable model. Groote [29] focused on TSSs that are *stratified*, which means that it is possible to define an appropriate weight function on the hypotheses and conclusions of the formal rules in a TSS. If a TSS is stratified, then it is complete. Our results also apply to complete (and so to stratified) TSSs.

4 Applications

Basically, Theorem 3.20 implies that a well-defined sum $T_0 \oplus T_1$ is a conservative extension of a TSS T_0 if two requirements are satisfied:

1. the formal rules in T_0 contain only source-dependent formal variables;
2. the sources of formal rules in T_1 are all fresh formal terms.

These two criteria, the first of which has been incorporated in the tool LATOS [33], are satisfied by most extensions of TSSs in the literature. We presented more liberal, and therefore more complicated, formulations of the two requirements in Theorem 3.20 on the forms of the formal rules in T_0 and T_1, in order to cover some cases of conservative extensions in the literature that do not satisfy one of the two criteria above.

1. The second requirement of Theorem 3.20 allows that formal variables in formal rules in T_0 are source-dependent *modulo a collection of sorts* $\mathcal{S}$, under the condition that for each $S \in \mathcal{S}$ there are no fresh actual terms of sort S.

   An example of the usefulness of this more liberal formulation is the specification language μCRL [31, 30], which consists of process algebra with data. The operational semantics of μCRL contains a formal rule for a sum construct $\Sigma(w.t)$, which simulates the behaviour of $t[d/w]$ for all possible data d:

   $$\frac{x^*[y^*/w] \xrightarrow{a} z^*}{\Sigma(w.x^*) \xrightarrow{a} z^*}$$

   where x^* and z^* are formal variables that range over a collection of process terms, and y^* is a formal variable and w an actual variable that range over some data domain, say of sort D. In this formal rule both y^* and z^* are not source-dependent (modulo $\emptyset$), but they are source-dependent modulo $\{D\}$, because y^* is of sort D. Hence, if the operational semantics of μCRL is extended with formal rules for a new process operator, say the state operator [3], but the data domains are not extended, then our format can be applied to conclude that such an extension is conservative.

2. The third requirement of Theorem 3.20 allows that a source of a formal rule r^* in T_1 is *not* a fresh formal term, under the condition that a fresh function symbol or fresh relation or predicate symbol occurs in a hypothesis of r^* that contains only original function symbols and source-dependent formal variables in its left-hand side.

   This generalization is useful in extensions of TSSs where the transition systems of original actual terms is extended. Examples of such extensions can be found in timed process algebra [42, 4]. In those two articles, untimed process algebra is extended with time, and original terms obtain the possibility to perform time steps. The operational semantics presented in those two articles contain formal rules such as

   $$\frac{x^* \xrightarrow{\sigma} x'^*}{x^* + y^* \xrightarrow{\sigma} x'^*}$$

where the source $x^* + y^*$, which denotes the alternative composition of processes x^* and y^*, is not fresh. However, the relation $\xrightarrow{\sigma}$, which expresses the execution of a time step, is fresh, and the left-hand side x^* of the hypothesis $x^* \xrightarrow{\sigma} x'^*$ in the formal rule above is a single source-dependent formal variable, so this rule does satisfy the more liberal third requirement in Theorem 3.20.

The extensions of TSSs described in [42, 4] are within the conservativity format described in this article, and, since these extensions do not contain binding constructs, also within the earlier format from [52].

Our conservativity format can be applied to extensions of operational semantics with binding constructs, such as in process algebra with time [16, 43, 21] or data [30, 31], where binding constructs enable to parametrize over the time or data domain, in process algebra with a recursive operator like the μ-construct [34, 54, 36, 48], in the π-calculus [40, 41, 47], and in the lazy λ-calculus [46, 35]. In the technical report version of this article [22] and in [23] it is shown how the conservativity format can also be applied in the realm of conditional rewriting. Finally, for applications of the conservativity format in the case of operational semantics without many-sortedness and binding mechanisms; see e.g. [7]. In the next two sections we give detailed applications of our conservativity result. The first section is devoted to a timed process algebra, while the second one focuses on the π-calculus.

4.1 Real Time ACP

We show how the conservativity result can be applied to real time ACP of Fokkink and Klusener [21], which is an adaptation of an earlier extension of ACP with real time by Baeten and Bergstra [2]. In [21] also the subalgebra real time BPA is considered, which does not take into account the communication operators of real time ACP, and it is claimed that real time ACP is a conservative extension of real time BPA, with a reference to the technical report version of this article [22]. Here, we present the technicalities to support this claim. Real time ACP is many-sorted, and contains a variable binding operator, called integration, so that previous conservativity formats could not be applied to its operational semantics.

First, we consider real time BPA, which consists of the following sorts and operators:

- *Atom* consists of a set of constants, referred to as the alphabet.
- *Time* also consists of a set of constants, and has the structure of an ordered field (see e.g. [15]). So in particular there are binary operators addition and multiplication on *Time*, which are commutative and associative.
- *Bound* consists of the terms defined by the BNF grammar

  $$b \quad ::= \quad t \mid x \mid b + b \mid t \cdot b$$

  where t represents an element of the ordered time domain, and x is an actual variable of sort *Time*.

- *Formula* consists of the boolean formulae defined by the BNF grammar

$$\phi \quad ::= \quad b < b' \mid \phi \wedge \phi \mid \neg\phi$$

  where b and b' represent bounds. Intuitively, $b < b'$ holds if b is smaller than b'.

- *Process* contains process terms that are defined by the BNF grammar

$$p \quad ::= \quad 0 \mid \int(a, x.\langle\phi, p\rangle) \mid p + p \mid b \gg p \mid \phi :\rightarrow p$$

  where 0 is a special constant, a a constant in the alphabet, x an actual time variable, ϕ a boolean formula, and b a bound. Process terms that do not contain free time variables specify behaviour according to the following intuitions:

  – 0 displays no behaviour;

  – $\int(a, x.\langle\phi, p\rangle)$ can execute action a at time t to evolve into $p[t/x]$, under the condition that the formula $\phi[t/x]$ is true;

  – $p + q$ executes the behaviour of either p or q;

  – $b \gg p$ consists of the behaviour of p after time t, with $b = t$;

  – $\phi :\rightarrow p$ equals either p, if formula ϕ is true, or 0, if formula ϕ is false.

Remark 4.1 Time is interpreted in an absolute way, that is, time numbers refer to some global clock. This contrasts with relative time, in which time numbers refer to the last moment in time that a previous action was executed.

The operational semantics for real time BPA is presented in Table 1, where b^*, b'^* are formal variables of sort *Bound*, ϕ^*, ψ^* are formal variables of sort *Formula*, p^*, p'^*, q^* are formal variables of sort *Process*, x is an actual variable of sort *Time*, a ranges over the constants in *Atom*, and finally s, t, u range over the constants in *Time*. The intuition behind the relations and predicates that are defined in Table 1 is as follows:

- $p \xrightarrow{a,t} p'$ expresses that process p can evolve into process p' by the execution of action a at time t;

- $U_t(p)$ holds if process p can execute an initial action after time t;

- $E_t(b)$ holds if bound b equals time number t;

- ϕT holds if formula ϕ is true.

The first predicate U_t is needed in the operational semantics for real time ACP. The last two predicates E_t and T are not present in [21], where the semantics of bounds and formulas are defined by means of equations. However, in order to apply the conservativity result to this setting, it is necessary to capture the semantics of bounds and formulas in formal rules, using the predicates E_t and T.

The formal variables in the formal rules in Table 1 are all source-dependent. As an example, we show that this is the case in the formal rule for the conditional construct

$$\frac{\phi^*[t/x]\,T}{\int(a,x.\langle\phi^*,p^*\rangle)\xrightarrow{a,t}t\gg p^*[t/x]} \qquad \frac{p^*\xrightarrow{a,t}p'^*}{p^*+q^*\xrightarrow{a,t}p'^*} \qquad \frac{p^*\xrightarrow{a,t}p'^*}{q^*+p^*\xrightarrow{a,t}p'^*}$$

$$\frac{(b^*<t)\,T \quad p^*\xrightarrow{a,t}p'^*}{b^*\gg p^*\xrightarrow{a,t}p'^*} \qquad \frac{\phi^*\,T \quad p^*\xrightarrow{a,t}p'^*}{\phi^*:\rightarrow p^*\xrightarrow{a,t}p'^*}$$

$$\frac{\phi^*[s/x] \quad (t<s)\,T}{U_t(\int(a,x.\langle\phi^*,p^*\rangle))} \qquad \frac{U_t(p^*)}{U_t(p^*+q^*)} \qquad \frac{U_t(p^*)}{U_t(q^*+p^*)}$$

$$\frac{U_t(p^*)}{U_t(b^*\gg p^*)} \qquad \frac{(t<b^*)\,T}{U_t(b^*\gg p^*)} \qquad \frac{\phi^*\,T \quad U_t(p^*)}{U_t(\phi^*:\rightarrow p^*)}$$

$$E_t(t) \qquad \frac{E_s(b^*) \quad E_t(b'^*)}{E_u(b^*+b'^*)}\ u=s+t \qquad \frac{E_s(b^*)}{E_u(t\cdot b^*)}\ u=t\cdot s$$

$$\frac{E_s(b^*) \quad E_t(b'^*)}{(b^*<b'^*)\,T}\ s<t \qquad \frac{\phi^*\,T \quad \psi^*\,T}{(\phi^*\wedge\psi^*)\,T} \qquad \frac{\phi^*\,\neg T}{(\neg\phi^*)\,T}$$

Table 1: Formal transition rules for real time BPA

$\phi^* :\rightarrow p^*$, which contains three formal variables: ϕ^*, p^* and p'^*. The formal variables ϕ^* and p^* in this formal rule occur in the source, so they are source-dependent. Moreover, since p^* is source-dependent and the formal rule contains the hypothesis $p^*\xrightarrow{a,t}p'^*$, the formal variable p'^* in this formal rule is also source-dependent.

Real time ACP is an extension of real time BPA; it introduces the binary communication operators $\|$ and $|$ and $\mathbin{\lfloor\!\lfloor}$. Thus, the syntax for the sorts *Atom* and *Time* and *Bound* and *Formula* and the relations and predicates remain the same, but the BNF grammar for the sort *Process* is extended with the three communication operators:

$$p \quad ::= \quad 0 \mid \int(a,x.\langle\phi,p\rangle) \mid p+p \mid b\gg p \mid \phi:\rightarrow p \mid p\|p \mid p|p \mid p\mathbin{\lfloor\!\lfloor}p$$

Note that this extension is well-defined (in the sense of Definition 3.1). We also assume a symmetric communication function γ between actions: $\gamma : \mathit{Atom}\times\mathit{Atom}\rightarrow\mathit{Atom}$. The intuition behind the communication operators is as follows:

- if process p can execute action a at time t to evolve into p', and process q can execute an initial action after time t, then $p\mathbin{\lfloor\!\lfloor}q$ can execute action a at time t to evolve into $p'\|q$;

- if process p can execute action a at time t to evolve into p', and process q can execute action a' at time t to evolve into q', then $p|q$ can execute the communication action $\gamma(a,a')$ at time t to evolve into $p'\|q'$;

- $p\|q$ combines the behaviours of $p\mathbin{\lfloor\!\lfloor}q$ and $q\mathbin{\lfloor\!\lfloor}p$ and $p|q$.

$$\frac{p^* \xrightarrow{a,t} p'^* \quad U_t(q^*)}{p^* \| q^* \xrightarrow{a,t} p'^* \| (t \gg q^*)} \qquad \frac{p^* \xrightarrow{a,t} p'^* \quad U_t(q^*)}{q^* \| p^* \xrightarrow{a,t} (t \gg q^*) \| p'^*} \qquad \frac{p^* \xrightarrow{a,t} p'^* \quad U_t(q^*)}{p^* \mathbin{⫽} q^* \xrightarrow{a,t} p'^* \| (t \gg q^*)}$$

$$\frac{p^* \xrightarrow{a,t} p'^* \quad q^* \xrightarrow{a',t} q'^*}{p^* \| q^* \xrightarrow{c,t} p'^* \| q'^*}\ \gamma(a,a') = c \qquad \frac{p^* \xrightarrow{a,t} p'^* \quad q^* \xrightarrow{a',t} q'^*}{p^* | q^* \xrightarrow{c,t} p'^* \| q'^*}\ \gamma(a,a') = c$$

$$\frac{U_t(p^*) \quad U_t(q^*)}{U_t(p^* \| q^*)} \qquad \frac{U_t(p^*) \quad U_t(q^*)}{U_t(p^* \mathbin{⫽} q^*)} \qquad \frac{U_t(p^*) \quad U_t(q^*)}{U_t(p^* | q^*)}$$

Table 2: Formal transition rules for real time ACP

These intuitions are formalized by means of the extra formal rules for real time ACP that are given in Table 2. It is easy to see that the sources of these formal rules are all fresh, since they all contain one of the communication operators. Hence, the third requirement of Theorem 3.20 is satisfied. Moreover, since the formal variables of the sort *Process* that occur in formal rules in Table 1 are all source-dependent, the second requirement of Theorem 3.20 is also satisfied. Hence, according to Theorem 3.20 real time ACP is a conservative extension of real time BPA.

Remark 4.2 The termination symbol 0, taken from CCS, is not present in [21], where processes can terminate successfully. We introduced the 0 here, because in the setting with 0 no extra formal rules for successful termination are needed, which reduces the number of formal rules in the operational semantics considerably. Furthermore, we excluded the deadlock δ, and the encapsulation operator ∂_H and its formal rules, which are present in [21]. Although the conservativity format can also handle these constructs, we preferred to leave them out, in order to keep the example as simple as possible.

4.2 The πI-Calculus

We show how the conservativity format can be applied to the πI-calculus from Sangiorgi [47], which is a subset of the full π-calculus. Basically, one could say that the πI-calculus is made out of CCS, combined with many-sortedness, variable binding and α-conversion. These extra features are outside the scope of previous conservativity formats. The formal transition rules for the π-calculus as defined in [40] satisfy our criteria too, so the conservativity result can be applied to that formalism just as well. However, we prefer πI over π here, because it has a simpler operational semantics, which allows to keep the exposition smooth.

We already encountered the πI-calculus, and its formal rule PRE, briefly in Example 2.15. We explain its syntax and semantics in more detail. Recall that there are two sorts *Port* and *Process*. Process terms are defined by the BNF grammar

$$p \quad ::= \quad 0 \mid x(y).p \mid \bar{x}(y).p \mid p + p \mid p|p \mid \nu\, y\, p$$

where 0 and p are terms of sort *Process*, and x and y are actual variables of sort *Port*. The occurrences of x in this grammar are free, while the occurrences of y are binders of p. As usual, $p + p'$ denotes the alternative composition and $p|p'$ the communication merge. The process $x(y).p$ sends, and the process $\bar{x}(y).p$ reads, port name y via port x and proceeds as p. In both expressions, the x is free, and the y is bound in p. Finally, $\nu\, y\, p$ expresses that the port name y is made local in p, that is, the y is bound in p.

$$\text{PRE}\quad x(y).v^* \xrightarrow{x(y)} v^* \qquad\qquad \text{SUM}\quad \frac{v^* \xrightarrow{x(y)} v'^*}{v^* + w^* \xrightarrow{x(y)} v'^*}$$

$$\text{PAR}\quad \frac{v^* \xrightarrow{x(y)} v'^* \quad \neg F_y(w^*)}{v^*|w^* \xrightarrow{x(y)} v'^*|w^*} \qquad\qquad \text{COM}\quad \frac{v^* \xrightarrow{x(y)} v'^* \quad w^* \xrightarrow{\bar{x}(y)} w'^*}{v^*|w^* \xrightarrow{\tau} \nu\, y\, (v'^*|w'^*)}$$

$$\text{RES}\quad \frac{v^* \xrightarrow{x(y)} v'^*}{\nu\, z\, v^* \xrightarrow{x(y)} \nu\, z\, v'^*}\quad z \notin \{x, y\}$$

$$F_x(x(y).v^*) \qquad\qquad F_x(\bar{x}(y).v^*)$$

$$\frac{F_z(v^*)}{F_z(x(y).v^*)}\quad z \neq y \qquad \frac{F_z(v^*)}{F_z(\bar{x}(y).v^*)}\quad z \neq y \qquad \frac{F_z(v^*)}{F_z(\nu\, y\, v^*)}\quad z \neq y$$

$$\frac{F_x(v^*)}{F_x(v^* + w^*)} \qquad \frac{F_x(v^*)}{F_x(w^* + v^*)} \qquad \frac{F_x(v^*)}{F_x(v^*|w^*)} \qquad \frac{F_x(v^*)}{F_x(w^*|v^*)}$$

Table 3: Operational semantics of the πI-calculus

The operational semantics of the πI-calculus is presented in Table 3, where $x,\ y,\ z$ are actual variables of sort *Port*, and $v^*,\ v'^*,\ w^*,\ w'^*$ are formal variables of sort *Process*. In order to keep Table 3 clean, the versions of PRE and SUM and PAR and RES with label $\bar{x}(y)$ instead of $x(y)$, and the symmetric versions of SUM and PAR and COM, have not been included. The x and y in the labels of the formal rules are free parameters.

The predicate F_y that is used in the negative hypothesis of PAR, holds for processes that contain free occurrences of the actual variable y. In most presentations of operational semantics for the π-calculus, a phrase "y not free in w^*" is added to PAR. However, in order to apply our conservativity result we need to give a more rigorous definition of this side condition. The inductive definition for F_y is captured by the nine formal rules at the lower end of Table 3.

The formal variables in the formal rules in Table 3 are all source-dependent. As an example, we show that this is the case for the formal rule COM. It says that if v^* sends port name y along port x, proceeding as v'^*, and if w^* reads port name y along port x, proceeding as w'^*, then their merge can communicate, proceeding as the merge of v'^*

and w'^*, in which the port name y is made local, i.e., is bound in both arguments. The formal variables in COM are all source-dependent:

- v^* and w^* occur in the source, so they are source-dependent,
- in the hypotheses $v^* \xrightarrow{x(y)} v'^*$ and $w^* \xrightarrow{\bar{x}(y)} w'^*$, the left-hand sides v^* and w^* are source-dependent, so their respective right-hand sides v'^* and w'^* are also source-dependent.

Since each formal rule in the operational semantics of the πI-calculus contains only source-dependent formal variables, Theorem 3.20 and implies that a well-defined sum $T_0 \oplus T_1$ is a conservative extension of the TSS T_0 for the πI-calculus if the sources of the formal rules in the extension T_1 are all fresh terms.

Remark 4.3 In the πI-calculus, port names are not processes, but data that are used to parametrize processes. Since processes and data are not distinguished in our setting, port names are considered to be processes too. This means that the conservativity result is slightly stronger than necessary, namely, that behaviour of both processes (interesting) and port names (not so interesting) is not influenced by the formal rules in the extension.

5 Conclusion

In this article we set up a formal framework to describe transition system specifications in the style of Plotkin. This framework has the power to express many-sortedness, general binding mechanisms and substitutions, among other notions such as negative hypotheses and unary predicates on terms. It can serve as a platform to prove general properties concerning transition system specifications.

We discussed one such result, known as conservativity. The conservativity theorem that we proved states under which circumstances the extension of a transition system specification with new formal rules does not affect the behaviour of the original terms. This subject is important because many existing operational semantics are extended with new features such as real time or mobility, and this should preferably be done conservatively.

References

[1] Allen, S.F., Constable, R.L., Howe, D.J., and Aitken, W.E. (1990), The semantics of reflected proof, *in* "Proceedings, 5th IEEE Symposium on Logic in Computer Science (LICS'90), Philadelphia", pp. 95–197, IEEE Computer Society Press.

[2] Baeten, J.C.M., and Bergstra, J.A. (1991), Real time process algebra, *Form. Asp. Comput.*, **3**(2), 142–188.

[3] Baeten, J.C.M., and Bergstra, J.A. (1991), Recursive process definitions with the state operator, *Theoret. Comput. Sci.*, **82**(2), 285–302.

[4] Baeten, J.C.M., and Bergstra, J.A. (1992), Discrete time process algebra, *in* "Proceedings, 3rd Conference on Concurrency Theory (CONCUR'92), Stony Brook", (W.R. Cleaveland, Ed.), pp. 401–420, Lecture Notes in Computer Science, Vol. 630, Springer-Verlag.

[5] Baeten, J.C.M., and Bergstra, J.A. (1996), Discrete time process algebra, *Form. Asp. of Comput.*, **8**(2), 188–208.

[6] Baeten, J.C.M., and Verhoef, C. (1993), A congruence theorem for structured operational semantics with predicates, *in* "Proceedings, 4th Conference on Concurrency Theory (CONCUR'93), Hildesheim", (E. Best, Ed.), pp. 477–492, Lecture Notes in Computer Science, Vol. 715, Springer-Verlag.

[7] Baeten, J.C.M., and Verhoef, C. (1995), Concrete process algebra, *in* "Handbook of Logic in Computer Science, Volume IV, Syntactical Methods", (S. Abramsky, D.M. Gabbay, and T.S.E. Maibaum, Eds.), pp. 149–268, Oxford University Press.

[8] Baeten, J.C.M., and Weijland, W.P. (1990), "Process Algebra", Cambridge Tracts in Theoretical Computer Science 18, Cambridge University Press.

[9] Barendregt, H.P. (1984), "The Lambda Calculus – Its Syntax and Semantics", Studies in Logic and the Foundations of Mathematics 103, North-Holland, Revised edition.

[10] Bernstein, K.L. (1998), "A Congruence Theorem for Structured Operational Semantics of Higher-Order Languages", *in* "Proceedings, 13th IEEE Symposium on Logic in Computer Science (LICS'98), Indianapolis", pp. 153–164. IEEE Computer Society Press, 1998.

[11] Bloom, B. (1995), Structural operational semantics for weak bisimulations, *Theoret. Comput. Sci.*, **146**(1/2), 25–68.

[12] Bloom, B., Istrail, S., and Meyer, A.R. (1995), Bisimulation can't be traced, *J. Assoc. Comput. Mech.*, **42**(1), 232–268.

[13] Bloom, B., and Vaandrager, F.W. (1995), SOS rule formats for parameterized and state-bearing processes, Unpublished manuscript.

[14] Bol, R.N., and Groote, J.F. (1996), The meaning of negative premises in transition system specifications, *J. Assoc. Comput. Mech.*, **43**(5), 863–914.

[15] Chang, C.C., and Keisler, H.J. (1990), "Model Theory", North-Holland.

[16] Chen, L. (1993), Axiomatising real-timed processes, *in* "Proceedings, 9th Conference on Mathematical Foundations of Programming Semantics (MFPS'93), New Orleans", (S. Brooks, M. Main, A. Melton, M. Mislove, and D. Schmidt, Eds.), pp. 215–229, Lecture Notes in Computer Science, Vol. 802, Springer-Verlag.

[17] Constable, R.L., *et al.* (1986), "Implementing Mathematics with the Nuprl Proof Development System", Prentice Hall.

[18] D'Argenio, P.R. (1995), A general conservative extension theorem in process algebras with inequalities, *in* "Proceedings, 2nd Workshop on the Algebra of Communicating Processes (ACP'95), Eindhoven", (A. Ponse, C. Verhoef, and S.F.M. van Vlijmen, Eds.), pp. 67–79, Report CS-95-14, Eindhoven University of Technology.

[19] D'Argenio, P.R., and Verhoef, C. (1997), A general conservative extension theorem in process algebras with inequalities, *Theoret. Comput. Sci.*, **177**(2), 351–380.

[20] Fokkink, W.J., and van Glabbeek, R.J. (1996), Ntyft/ntyxt rules reduce to ntree rules, *Inform. Comput.*, **126**(1), 1–10.

[21] Fokkink, W.J., and Klusener, A.S. (1995), An effective axiomatization for real time ACP, *Inform. Comput.*, **122**(2), 286–299.

[22] Fokkink, W.J., and Verhoef, C. (1995), "A Conservative Look at Term Deduction Systems with Variable Binding", Logic Group Preprint Series 140, Utrecht University, Available at http://www.phil.uu.nl.

[23] Fokkink, W.J., and Verhoef, C. (1998), "Conservative Extension in Positive/Negative Conditional Term Rewriting with Applications to Software Renovation Factories", Report P9802, University of Amsterdam. Available at http://adam.wins.uva.nl/ x.

[24] van Gelder, A., Ross, K., and Schlipf, J.S. (1991), The well-founded semantics for general logic programs, *J. Assoc. Comput. Mech.*, **38**(3), 620–650.

[25] Gelfond, M., and Lifschitz, V. (1988), The stable model semantics for logic programming, *in* "Proceedings, 5th Conference on Logic Programming", pp. 1070–1080, MIT Press.

[26] van Glabbeek, R.J. (1993), Full abstraction in structural operational semantics, *in* "Proceedings, 3rd Conference on Algebraic Methodology and Software Technology (AMAST'93), Enschede", (M. Nivat, C. Rattray, T. Rus, and G. Scollo, Eds.) pp. 77–84, Workshops in Computing, Springer-Verlag.

[27] van Glabbeek, R.J. (1995), "The Meaning of Negative Premises in Transition System Specifications II", Report STAN-CS-TN-95-16, Stanford University. Available at http://theory.stanford.edu/ rvg/.

[28] van Glabbeek, R.J. (1996), The meaning of negative premises in transition system specifications II, Extended abstract *in* "Proceedings, 23rd Colloquium on Automata, Languages and Programming (ICALP'96), Paderborn", (F. Meyer auf der Heide and B. Monien, Eds.), pp. 502–513, Lecture Notes in Computer Science, Vol. 1099, Springer-Verlag.

[29] Groote, J.F. (1993), Transition system specifications with negative premises, *Theoret. Comput. Sci.*, **118**(2), 263–299.

[30] Groote, J.F., and Ponse, A. (1994), Proof theory for μCRL: a language for processes with data, *in* "Proceedings, Workshop on Semantics of Specification Languages", (D.J. Andrews, J.F. Groote, and C.A. Middelburg, Eds.), pp. 232–251, Workshops in Computing, Springer-Verlag.

[31] Groote, J.F., and Ponse, A. (1995), Syntax and semantics of μCRL, *in* "Proceedings, 1st Workshop on the Algebra of Communicating Processes (ACP'94), Utrecht", pp. 26–62, Workshops in Computing, Springer-Verlag.

[32] Groote, J.F., and Vaandrager, F.W. (1992), Structured operational semantics and bisimulation as a congruence, *Inform. Comput.*, **100**(2), 202–260.

[33] Hartel, P.H., (1997), "LATOS – A Lightweight Animation Tool for Operational Semantics", Report DSSE-TR-97-1, University of Southampton, Available at http://www.dsse.ecs.soton.ac.uk.

[34] Hennessy, M. (1988), "Algebraic Theory of Processes", MIT Press.

[35] Hennessy, M. (1994), A fully abstract denotational model for higher-order processes, *Inform. Comput.*, **112**(1), 55–95.

[36] Hennessy, M., and Ingólfsdóttir, A. (1993), A theory of communicating processes with value passing, *Inform. Comput.*, **107**(2), 202–236.

[37] Hoare, C.A.R. (1985), "Communicating Sequential Processes", Prentice Hall.

[38] Howe, D.J. (1996), Proving congruence of bisimulation in functional programming languages, *Inform. Comput.*, **124**(2), 103–112.

[39] Milner, R. (1989), "Communication and Concurrency", Prentice Hall.

[40] Milner, R., Parrow, J., and Walker, D. (1991), Modal logics for mobile processes, *in* "Proceedings, 2nd Conference on Concurrency Theory (CONCUR'91), Amsterdam", (J.C.M. Baeten and J.F. Groote, Eds.), pp. 45–60, Lecture Notes in Computer Science, Vol. 527, Springer-Verlag.

[41] Milner, R., Parrow, J., and Walker, D. (1992), A calculus of mobile processes, part I + II, *Inform. Comput.*, **100**(1), 1–77.

[42] Moller, F., and Tofts, C. (1990), A temporal calculus of communicating systems, *in* "Proceedings, 1st Conference on Concurrency Theory (CONCUR'90), Amsterdam", (J.C.M. Baeten and J.W. Klop, Eds.), pp. 401–415, Lecture Notes in Computer Science, Vol. 458, Springer-Verlag.

[43] Nicollin, X., and Sifakis, J. (1994), The algebra of timed processes, ATP: theory and application, *Inform. Comput.*, **114**(1), 131–178.

[44] Plotkin, G.D. (1981), "A Structural Approach to Operational Semantics", Report DAIMI FN-19, Aarhus University.

[45] Przymusinski, T.C. (1990), The well-founded semantics coincides with the three-valued stable semantics, *Fundam. Informat.*, **13**(4), 445–463.

[46] Sangiorgi, D. (1994), The lazy lambda calculus in a concurrency scenario, *Inform. Comput.*, **111**(1), 120–153.

[47] Sangiorgi, D. (1995), πI: A symmetric calculus based on internal mobility, *in* "Proceedings, 6th Conference on Theory and Practice of Software Development (TAPSOFT'95), Aarhus", (P.D. Mosses and M. Nielsen and M.I. Schwartzbach, Eds.), pp. 172–186, Lecture Notes in Computer Science, Vol. 915, Springer-Verlag.

[48] Schneider, S.A. (1995), An operational semantics for timed CSP, *Inform. Comput.*, **116**(2), 193–213.

[49] de Simone, R. (1985), Higher-level synchronising devices in MEIJE–SCCS, *Theoret. Comput. Sci.*, **37**(3), 245–267.

[50] Stoughton, A. (1988), Substitution revisited, *Theoret. Comput. Sci.*, **59**(3), 317–325.

[51] Tarski, A. (1955), A lattice theoretical fixed point theorem and its applications, *Pacific J. of Mathem.*, **5**, 285–309.

[52] Verhoef, C. (1994), A general conservative extension theorem in process algebra, *in* "Proceedings, IFIP Working Conference on Programming Concepts, Methods and Calculi (PROCOMET'94), San Miniato", (E.-R. Olderog, Ed.), pp. 149–168, IFIP Transactions A-56, Elsevier.

[53] Verhoef, C. (1995), A congruence theorem for structured operational semantics with predicates and negative premises, *Nordic J. Comput.*, **2**(2), 274–302.

[54] Wang, Y. (1991), CCS + time = an interleaving model for real time systems, *in* "Proceedings, 18th Colloquium on Automata, Languages and Programming (ICALP'91), Madrid", (J. Leach Albert, B. Monien, and M. Rodríguez, Eds.), pp. 217–228, Lecture Notes in Computer Science, Vol. 510, Springer-Verlag.

[55] Watt, D.A. (1990), "Programming Concepts and Paradigms", Prentice Hall.