A Conservative Look at Term Deduction Systems with Variable Binding

Wan Fokkink
Department of Philosophy
Utrecht University
Heidelberglaan 8, 3584 CS Utrecht, The Netherlands
fokkink@phil.ruu.nl

Chris Verhoef
Department of Mathematics and Computing Science
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
chrisv@win.tue.nl

Abstract

We set up a formal framework to describe term deduction systems, such as transition system specifications in the style of Plotkin, and conditional term rewriting systems. This framework has the power to express many-sortedness, general binding mechanisms and substitutions, among other notions such as negative premises 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 rules does not affect the behaviour of the original terms.

Furthermore, we show how general theorems in structured operational semantics can be transformed into results in conditional term rewriting. We apply this approach to the conservativity theorem, which yields a result that is useful in the field of abstract data types.

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 [33]. Given a set of states, the transitions between these states are obtained inductively from a transition system specification (TSS), which consists of transition rules.

Deducing desirable properties for transition systems generated by some TSS is often a technical and mechanic labour. Therefore, in recent years several general theories for TSSs have been developed, for instance to discover which TSSs satisfy a certain congruence property [37, 11, 22, 21, 13, 4, 40, 16, 24, 10], or to study the meaning of negative premises [21, 13, 20], or to find which extensions of TSSs are operationally conservative [22, 21, 13, 39]. Our paper is devoted to this last topic.

Over and over again, process theories such as CCS [28], CSP [23] and ACP [8] have been extended with new features, and the original TSSs, which provide the semantics for these process algebras, were extended with rules to describe these features. 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 [22] were the first to propose (in theorem 7.6) syntactic restrictions for the original TSS and for its extension, which automatically yield that the extension is operationally conservative. The restrictions are: all rules must be 'tyft/tyxt', and the original rules must be 'pure' and 'well-founded' (see [22] for the definitions), and the rules in the extension must contain some new operator in their source, i.e., in the left-hand side of their conclusion. Groote [21] adapted this conservativity format to the setting with negative hypotheses. Bol and Groote [13] showed that the tyft/tyxt restriction can be omitted.

Finally, Verhoef [39] proposed more general syntactic criteria which ensure operational conservativity. Verhoef's criteria allow, under certain conditions, that a 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 [31], and BPA with discrete time from Baeten and Bergstra [3]. (In the current version of BPA with discrete time, the operational semantics has been adapted in such a way that the extension with discrete time is no longer operationally conservative over BPA.)

In many practical cases, the format from [39] cannot yet be applied, due to the use of a typed 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 paper proposes a generalization of the conservativity format to transition rules which may contain types and a variable binding mechanism. The syntactic requirements from Verhoef's format are generalized to this setting. These generalizations make heavy use of two different notions of free variables in terms; these notions will be introduced in due course. The syntactic criteria 'pure' and 'well-founded' on original rules will be relaxed to a more natural formulation, which we will call source-dependency.

We found that 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 will 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.

There is a strong link between the world of structured operational semantics and that of conditional term rewriting. Both fields deal with what can be viewed as 'term deduction systems'. Terms are built from a set of function symbols, and binary relations on terms (transitions versus rewrite steps) are defined by means of proof rules (transition

rules versus conditional rewrite rules). Such a rule, together with the validity, or non-validity, of a number of relations between terms, may imply the validity of another relation between terms. Hence, theorems in structured operational semantics also apply to the field of conditional rewriting. The link was already noted, but not exploited, by Groote and Vaandrager [22] (in example 3.5). We show that our conservativity results constitute useful lemmas in the field of conditional rewriting. Most notably, the results are applicable in the field of abstract data types, which are often specified as modules of conditional term rewriting systems.

Acknowledgements. We have benefited from technical discussions with Bard Bloom, Pedro D'Argenio, Arie van Deursen, Rob van Glabbeek, Douglas Howe, Davide Sangiorgi, and Frits Vaandrager.

2 The Formal Framework

In this section we recall some notions concerning general theory of structured operational semantics, and we introduce some new matters, interspersed with examples. We incorporate the notions of negative hypotheses from Groote [21] and predicates from Baeten and Verhoef [4]. We define a framework in which it is possible to express binding mechanisms and substitutions. To that end we will introduce two different kinds of terms: actual ones and formal ones. Before we continue with the definitions, we give some intuition about those signatures and the framework.

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, [41]. We will do the same in this paper: 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 we will discuss transition rules on a conceptual level, we will 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.

We illustrate with the (nonsense) transition rule below that it is useful to make a notational distinction between actual and formal variables.

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

Application of a substitution σ to this transition rule yields

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

For instance, if $\sigma(x) = c$ 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 $y[x/x]$ is not a substitution (for then it would equal y), but a syntactic construct with a suggestive form. We will call it a *substitution harness*. Only after application of a substitution σ, the result $\sigma(y)[\sigma(x)/x]$ 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 following form:

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

where x and x^* are unrelated.

The distinction of formal variables in structured operational semantics with variable binding was also propagated independently by Sangiorgi [35] and by Howe [24]. There, they are called 'meta-variables'.

Now that we have an idea of the framework, we will 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, so we have chosen to describe these mechanisms as general as possible, in a notationally convenient way. We use a notational approach based on [2]; it is the notation for terms in the Nuprl proof development system, see [14]. Our choice for the Nuprl notation, instead of for example the λ-calculus [6], 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, $\vec{O}$ will denote a sequence $O_1 \ldots O_k$, and $\vec{O}_i$ a sequence $O_{i1} \ldots O_{ik}$, with $k \geq 0$. We note here that we do not need this k to be a finite number; in fact, our results are also valid if k is an arbitrary cardinal number. However, since we did not find any applications for infinite cardinals and since we want to focus the discussion on more important and less trivial matters we refrain from this generalization.

Definition 2.1 *A (many-sorted) signature Σ consists of a non-empty set of sorts, a set $\mathcal{V}$ of sorted actual variables $v, w, x, y, z, \ldots$, 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 *$s, t, \ldots$ over Σ is defined as the least set satisfying:*

- *each actual variable from $\mathcal{V}$ is in $\mathbb{T}(\Sigma)$,*
- *for each function $f : \vec{S}_1.S_1 \times \cdots \times \vec{S}_n.S_n \to S$, $f(\vec{x}_1.t_1, \ldots, \vec{x}_n.t_n)$ 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 $x_{i1} \ldots x_{im_i}$, with x_{ij} of sort S_{ij}.*

We say that the actual variables $\vec{x}_i$ are 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,*
- *if x occurs* free *in t_i, and $x \notin \{\vec{x}_i\}$, then x occurs* free *in $f(\vec{x}_1.t_1, \ldots, \vec{x}_n.t_n)$.*

As usual, 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 $p, q, \ldots$ over Σ.*

The notion of a substitution is also defined as expected.

Definition 2.4 *An* actual substitution *is a sort preserving mapping $\sigma : \mathcal{V} \to \mathbb{T}(\Sigma)$, where sort preserving means that x and $\sigma(x)$ are always of the same sort. A substitution extends to a mapping from terms to terms as usual; the term $\sigma(t)$ is obtained by replacing each free occurrence of a 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 will be called explicit *actual substitutions (as opposed to* implicit *actual substitutions σ).*

In the definition of substitutions on actual terms there is a well-known complication. Namely, consider a term $\sigma(t)$, and let x occur free in t. After x in t has been replaced by $\sigma(x)$, variables y that occur in $\sigma(x)$ are suddenly bound in 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 variables. In the sequel, actual terms are considered modulo α-conversion, and when a substitution is applied, bound variables are renamed. Stoughton [38] presented a nice treatment of this technique.

Remark 2.5 Bloom and Vaandrager [12] develop a framework for transition rules with types and a binding mechanism, in which they make a clear distinction between sorts for processes and sorts for data. We have chosen not to adopt this distinction, since it is not of interest for the question whether an extension of transition rules influences the behaviour of original process terms. We consider data as processes that do not display any behaviour.

2.2 The formal world

We argued that it is often a good idea to distinguish between formal and actual variables, when discussing transition rules with variable bindings and substitutions on an abstract level. This is also convenient in order to distinguish the variables to which a substitution applies. 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 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* $f : \vec{S}_1.S_1 \times \cdots \times \vec{S}_n.S_n \to S$, $f(\vec{x}_1.t_1^*, \ldots, \vec{x}_n.t_n^*)$ *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 $y^*[z^*/x]$, which evaluates to the actual term a after application of a formal substitution σ^* with $\sigma^*(y^*) = x$ and $\sigma^*(z^*) = a$. Namely, the implicit formal substitution σ^* turns the substitution harness $y^*[z^*/x]$ into the actual term $x[a/x]$, where $\_[a/x]$ is an explicit actual substitution, which evaluates to a.

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 actual terms.
- Formal substitutions σ^* are mappings from *formal* variables to *actual* terms.

- A substitution harness $t^*[s^*/x]$ is *not* a substitution, but a piece of syntax with a suggestive form. If we apply a formal substitution σ^* to it, 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 actual and formal substitutions are used to move from transition rules to a proof tree.

2.3 Actual and formal transition rules

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

Before we present the formal 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 will serve 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 can be expressed in the following transition rule, which we will call the μ-rule:

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

We 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. For the μ-construct we have

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

where $a\_$ is the well-known action prefix operator from CCS. The μ-rule is a recipe to achieve transitions like the one above. We derive this transition from the μ-rule together with the well-known rule for the prefix operator: $aw^* \xrightarrow{a} w^*$. After application of a 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 rule for the prefix operator, with $\mu x.ax$ for w^*, we may conclude that the σ^* instantiation of the conclusion of the μ-rule is valid: $\mu x.ax \xrightarrow{a} \mu x.ax$.

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

Definition 2.10 *Let $t_0, \ldots, t_n$ be terms over some signature.*

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

- *For R a predicate, the expression $t_0 R(t_1, \ldots, t_{n-1})$ is a* positive transition.

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

A transition is called closed *if it involves only closed actual terms.*

We allow the possibility to attach terms to relations and predicates, because of the fact that nowadays many transition rules use parametrized labels. We will see an example of such a parametrized label, in the setting of the π-calculus from Milner, Parrow and Walker [30], in example 2.14.

Definition 2.11 *An* actual (transition) rule *is an expression of the form H/c, where H is a collection of positive and negative transitions over an actual signature, and c is a positive transition over an actual signature.*

Example 2.12 An example of an actual rule that we met already is

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

It can be deduced from the μ-rule, see example 2.9, 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 we meet in the literature; they are the recipes that allow us to deduce the transition relation. We recall that it is our aim to make this recipe practice (more) precise, in order to be able to discuss properties of such rules on an abstract level.

Definition 2.13 *A* formal (transition) rule *is an expression of the form H^*/c^*, where*

- *H^* is a collection* hypotheses *of the form $t_0^* R(t_1^*, \ldots, t_{n-1}^*) t_n^*$ and $t_0^* R(t_1^*, \ldots, t_{n-1}^*)$ and $t_0^* \neg R(t_1^*, \ldots, t_{n-1}^*)$,*

- *c^* is the* conclusion *of the form $t_0^* R(t_1^*, \ldots, t_{n-1}^*) t_n^*$ or $t_0^* R(t_1^*, \ldots, t_{n-1}^*)$,*

where $t_0^*, \ldots, t_n^*$ are formal terms.
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.

Example 2.14 Assume two sorts *Port* of port names and *Process* of processes. For actual variables x and y of sort *Port* we have the following transition rule:

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

where v^* is a formal variable of sort *Process*. The rule PRE expresses that term $x(y).p$ sends port name y via port x, and proceeds as 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 paper 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 occurrence of the actual variable w of sort *Port*. 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 $x(y).t$.

In example 2.9 we already saw that a TSS is used to prove that certain transitions hold. Now we give the precise definition of a proof from a TSS.

Definition 2.15 *A* proof *from a TSS T of an actual rule H/c consists of an upwardly branching tree in which all upward paths are finite, where the nodes of the tree are labelled by transitions such that:*

- *the root has label c,*
- *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 $l \in H$,*
  2. *or K/l is a formal substitution instance of a formal rule in T.*

A proof is called closed *if all its labels are closed transitions.*

Example 2.16 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 rule for prefixing from CCS and the μ-rule. This proof is depicted in Figure 1.

Remark 2.17 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 2.14, where the transition $x(y).t \xrightarrow{x(w)} t[w/y]$, with $w \neq y$ not free in t, could only be proved by α-conversion of $x(y).t$ to $x(w).t[w/y]$.

2.4 On substitution instances of proofs

Due to the fact that substitution harnesses are allowed to occur in formal rules, the actual substitution instance of a proof need not be a proof again. This complication may arise in case the proof involves formal rules that contain two kinds of occurrences of one formal variable: one in a free context and the other in a bound context. We give an example.

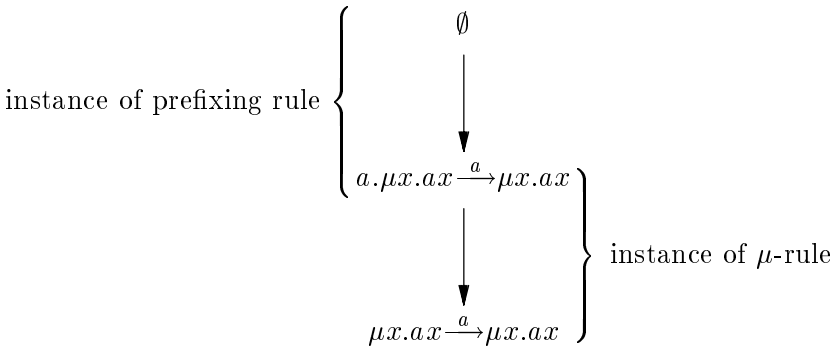

Figure 1: A proof for $\mu x.ax \xrightarrow{a} \mu x.ax$

Example 2.18 Let a and b be constants, and assume a predicate $\downarrow$. Consider the TSS that consists of the following formal rules:

$$a\downarrow \qquad \frac{y^*[a/x]\downarrow}{y^*\downarrow} \qquad \frac{y^*\downarrow}{y^*[b/x]\downarrow}$$

Note that in the last two rules, the formal variable y^* occurs both in a free and in a bound context.

By substituting x for y^* in the last two formal rules, we obtain the actual rules $a\downarrow / x\downarrow$ and $x\downarrow / b\downarrow$ respectively. Hence, the transition $b\downarrow$ is provable from the TSS. However, it is not hard to see that there does not exist a closed proof for $b\downarrow$.

2.5 Stable models

Up to now we have ignored the presence of negative hypotheses in the hypotheses of the rules in a TSS. It is well-known that when there are negative hypotheses around it is no longer straightforward to define a sensible transition relation. We will use the notion of a *stable model* for a TSS, which stems from Gelfond and Lifschitz [18], in the setting of logic programming, and which was adapted to structured operational semantics by Bol and Groote [13]. See Van Glabbeek [20] for an overview of other possibilities to give meaning to negative hypotheses in a TSS. We follow the approach in [20] and we adapt it to the present situation with predicates around.

Definition 2.19 *A collection of negative transitions H* holds *for a set of positive transitions $\mathcal{M}$, denoted by $\mathcal{M} \models H$, if for each $t\neg R(t_1,\ldots,t_{n-1}) \in H$ we have*

- *either $t_0 R(t_1,\ldots,t_{n-1})t_n \notin \mathcal{M}$ for all actual terms t_n if R is a relation.*
- *or $t_0 R(t_1,\ldots,t_{n-1}) \notin \mathcal{M}$ if R is a predicate.*

Definition 2.20 *A collection of positive transitions $\mathcal{M}$ is a* stable model *for a TSS T, if the elements of $\mathcal{M}$ are exactly those positive transitions c for which there exists an actual rule H/c such that:*

- *there is a proof from T for H/c,*
- *H contains only negative transitions,*
- *$\mathcal{M} \models H$.*

Similarly, we can define the notion of a *closed* stable model for a TSS.

Definition 2.21 *A collection of* closed *positive transitions $\mathcal{M}$ is a* closed stable model *for a TSS T, if the elements of $\mathcal{M}$ are exactly those* closed *positive transitions c for which there exists an actual rule H/c such that:*

- *there is a* closed *proof from T for H/c,*
- *H contains only* closed *negative transitions,*
- *$\mathcal{M} \models H$.*

There exist TSSs which allow several (closed) stable models.

Example 2.22 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 two stable models, namely $\{aR\}$ and $\{bR\}$.

The following example shows that the closed transitions in a stable model of a TSS do not always make out a closed stable model of this TSS.

Example 2.23 The TSS in example 2.18 allows one stable model $\{aR, bR\}$ and one closed stable model $\{aR\}$. The closed transition bR is not present in the closed stable model since it cannot be deduced by a closed proof.

We will prove a conservativity result both for stable models and for closed stable models. Due to the fact that the substitution instance of a proof from a TSS is not always a proof from this TSS, the conservative extension theorem for stable models does not immediately imply the same result for closed stable models. Fortunately, the conservative extension theorems for the open and for the closed case can be proved in exactly the same way, the only difference being that the proofs involve open and closed actual terms respectively.

Remark 2.24 Van Glabbeek [20] argues that the best way to give meaning to TSSs with negative hypotheses is through the notion of *completeness*. For this purpose, the notion of provability is extended in order to allow the derivation of negative transitions. Then, a TSS is said to be *complete* if for each closed transition $p \xrightarrow{a} p'$, the TSS can prove either $p \xrightarrow{a} p'$ or its negation $p \not\xrightarrow{a} p'$.

Our conservativity result for closed stable models applies to complete TSSs as well. Namely, if a TSS is complete, then it allows a unique closed stable model, which consists of the closed transitions that are provable from the TSS, see [20].

3 A General Conservative Extension Theorem

In this section, we present our theorem concerning conservative extensions. First, we define in a precise way what is a conservative extension; we distinguish between closed and open conservative extensions. Then a string of technical definitions will lead to the formulation of the main theorem.

3.1 Well-defined sums

In order to be able to combine two TSSs, we need to know that the function symbols in the intersection of their signatures have the same functionality in both signatures. Furthermore, if a relation or predicate symbol occurs in the two TSSs, then it should 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* Σ_0 *and* Σ_1 *be signatures. Their sum (or union)* $\Sigma_0 \oplus \Sigma_1$ *is* well-defined *if each function symbol and each variable in* $\Sigma_0 \cap \Sigma_1$ *has the same functionality in both signatures.*
- *Let* $\mathcal{D}_0$ *and* $\mathcal{D}_1$ *be sets of relations and predicates symbols. The sum (or union)* $\mathcal{D}_0 \oplus \mathcal{D}_1$ *is* well-defined *if each element in* $\mathcal{D}_0 \cap \mathcal{D}_1$ *is either a relation or a predicate in both collections.*
- *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 both* $\Sigma_0 \oplus \Sigma_1$ *and* $\mathcal{D}_0 \oplus \mathcal{D}_1$ *are well-defined.*

3.2 Conservative extension

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.

Definition 3.2 $T_0 \oplus T_1$ *is an* open (operationally) conservative extension *of* T_0 *if for each stable model* $\mathcal{M}$ *for* $T_0 \oplus T_1$*, the collection*

$$\{t_0 R(t_1, \ldots, t_{n-1}) t_n,\ t_0 R(t_1, \ldots, t_{n-1}) \in \mathcal{M} \mid t_0 \in \mathbb{T}(\Sigma_0)\}$$

is a stable model for T_0*.*

Definition 3.3 $T_0 \oplus T_1$ *is a* closed (operationally) conservative extension *of* T_0 *if for each* closed *stable model* $\mathcal{M}$ *for* $T_0 \oplus T_1$*, the collection*

$$\{p_0 R(p_1, \ldots, p_{n-1}) p_n,\ p_0 R(p_1, \ldots, p_{n-1}) \in \mathcal{M} \mid p_0 \in \mathcal{T}(\Sigma_0)\}$$

is a closed *stable model for* T_0*.*

Before we can formulate under what conditions $T_0 \oplus T_1$ is both a closed and an open conservative extension of T_0, first we need to present several auxiliary definitions.

3.3 New formal terms and new relations

A formal term in $\mathbb{F}(\Sigma_1)$ is called *new* if it incorporates a function symbol from $\Sigma_1 \backslash \Sigma_0$ outside its substitution harnesses. This is stated more precisely in the definition below.

Definition 3.4 *The formal terms in $\mathbb{F}(\Sigma_1)$ that are new are defined inductively as follows:*

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

Example 3.5 *Let $\Sigma_0 = \{a, f\}$ and $\Sigma_1 = \{b\}$, where a, b are constants and f is of arity one. Then $f(x.b[a/y])$ is new, but $f(x.a[b/y])$ is not new.*

Definition 3.4 is motivated by the following observation.

Lemma 3.6 *t^* is new $\Rightarrow \sigma^*(t^*) \notin \mathbb{T}(\Sigma_0)$.*

Proof. By induction on the size of t^*.

Definition 3.7 *Relations and predicates are called* new *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 3.8 *The collections $FV(t^*)$ are defined inductively as follows.*

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

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

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

Example 3.9 $FV(f(v.x^*[y^*/w])) = \{x^*, y^*\}$.

Lemma 3.10 *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 on the size of t^*.

The converse of lemma 3.10 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 3.11 Let $\Sigma_0 = \{a\}$ and $\Sigma_1 = \{b\}$, where a and b are constants, and let $\sigma^*(x^*) = a$ and $\sigma^*(y^*) = b$. Then $\sigma^*(x^*[y^*/z]) = a \in \mathbb{T}(\Sigma_0)$, but $\sigma^*(y^*) = b \notin \mathbb{T}(\Sigma_0)$.

In order to obtain a result converse to lemma 3.10, 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 3.12 *The collections $EV(t^*)$ are defined inductively as follows.*

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

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

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

Example 3.13 $EV(f(v.x^*[y^*/w])) = \{x^*\}$.

The definition of $EV(t^*)$ is motivated by the following observation, which is the converse of lemma 3.10, with FV replaced by EV.

Lemma 3.14 $\sigma^*(t^*) \in \mathbb{T}(\Sigma_0) \;\Rightarrow\; \sigma^*(x^*) \in \mathbb{T}(\Sigma_0)$ *for all* $x^* \in EV(t^*)$.

Proof. By induction on the size of t^*.

3.5 Source-dependency

Definition 3.15 *The formal term at the left-hand side of the conclusion of a formal rule is called the* source *of the formal rule.*

We adapt the syntactic criterion 'pure and well-founded' for transition rules from Groote and Vaandrager [22] to a more liberal form, which we call *source-dependency*. This definition uses the two distinct notions FV and EV of formal variables in formal terms. In the setting without variable bindings, this notion was also discovered independently by Van Glabbeek [19] and by Howe [24].

Definition 3.16 *For a formal rule r^*, its collection of* source-dependent *formal variables $SV(r^*)$ is defined inductively as follows.*

- *If t^* is the source of r^*, then $EV(t^*) \subseteq SV(r^*)$.*
- *If $t_0^* R(t_1^*, \ldots, t_{n-1}^*) t_n^*$ is a hypothesis of r^* and $FV(t_0^*) \subseteq SV(r^*)$, then $EV(t_i^*) \subseteq SV(r^*)$ for $i = 1, \ldots, n$.*
- *If $t_0^* R(t_1^*, \ldots, t_{n-1}^*)$ is a hypothesis of r^* and $FV(t_0^*) \subseteq SV(r^*)$, then $EV(t_i^*) \subseteq SV(r^*)$ for $i = 1, \ldots, n-1$.*

r^* is source-dependent *if the formal variables that occur in r^* are all in $SV(r^*)$.*

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

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

The variables in the μ-rule are all source-dependent. Namely, y^* and z^* are the only formal variables that occur in the μ-rule. Since $EV(\mu x.y^*) = \{y^*\}$, we find that y^* is source-dependent. Since

$$FV(y^*[\mu x.y^*/x]) \;=\; FV(y^*) \cup FV(\mu x.y^*) \;=\; \{y^*\},$$

and since the μ-rule contains a premise $y^*[\mu x.y^*/x] \xrightarrow{a} z^*$, and since $EV(z^*) = \{z^*\}$, it follows that z^* is source-dependent.

We will see later on that source-dependency is an essential ingredient of the conservativity theorem. Namely, in order to conclude that an extended TSS is operationally conservative over an original TSS, it is necessary that the rules in the original TSS are source-dependent. In practical cases, this criterion is sometimes neglected. For example, Nicollin and Sifakis [32] consider an extended TSS in which each rule in the extension contains a new operator in its source, and from this fact they conclude that the extension is operationally conservative. In general however, this characteristic is not sufficient, as is shown in the next example.

Example 3.18 Let $\Sigma_0 = \{a\}$ and $\Sigma_1 = \{b\}$, where a and b are constants, and let $\downarrow$ be a predicate. Consider the TSS over Σ_0 that consists of the rule $x^*\downarrow \,/ a\downarrow$. Note that the formal variable x^* in this rule is not source-dependent. Extend this TSS with the rule $b\downarrow$, which contains the new constant b in its source. Then $a\downarrow$ holds in the extended TSS, but not in the original one, so this extension is not operationally conservative.

3.6 The formal rule $\rho(r^*)$

Definition 3.19 *For each formal rule r^* in $T_0 \oplus T_1$, $\rho(r^*)$ denotes the formal rule that consists of the conclusion from r^*, together with those hypotheses from r^* for which the term at the left-hand side is in $\mathbb{F}(\Sigma_0)$.*

Example 3.20 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 rule

$$\frac{a\downarrow \qquad b\downarrow}{b\uparrow}$$

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 3.21 formulates sufficient criteria for $T_0 \oplus T_1$ to be a closed conservative extension of T_0. As a corollary we will find that the same criteria ensure that $T_0 \oplus T_1$ is an open conservative extension of T_0.

Theorem 3.21 *Under the following conditions, $T_0 \oplus T_1$ is a closed conservative extension of T_0:*

1. *T_0 is source-dependent,*
2. *for each $r^* \in T_1$,*
   - *either the source of r^* is new,*
   - *or r^* has a hypothesis of the form $t_0^* R(t_1^*, \ldots, t_{n-1}^*)t_n^*$ or $t_0^* R(t_1^*, \ldots, t_{n-1}^*)$, where*
     - *$t_0^* \in \mathbb{F}(\Sigma_0)$,*
     - *$FV(t_0^*) \subseteq SV(\rho(r^*))$,*
     - *R or one of the terms $t_1^*, \ldots, t_n^*$ is new.*

In the proof of this conservativity theorem we will apply induction on the *source-distance* of a formal variable x^* in a formal rule r^*, being the minimal number of steps it takes to deduce that x^* is source-dependent in r^*.

Definition 3.22 *Assume a formal rule r^*. For a formal variable $x^* \in SV(r^*)$, its* source distance *$sd(r^*, x^*)$ in r^* is defined as follows.*

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

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

Proof of theorem 3.21. Fix a closed stable model $\mathcal{M}$ for $T_0 \oplus T_1$. We show that

$$\mathcal{N} = \{p_0 R(p_1, \ldots, p_{n-1})p_n,\ p_0 R(p_1, \ldots, p_{n-1}) \in \mathcal{M} \mid p_0 \in \mathcal{T}(\Sigma_0)\}$$

is a closed stable model for T_0.

1. Assume that there is a closed proof from T_0 for a closed actual rule H/c, where H contains only negative transitions, and $\mathcal{N} \models H$. We show that $c \in \mathcal{N}$.

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

   Consider a $p_0 \neg R(p_1, \ldots, p_{n-1})$ in H. Since $\mathcal{N} \models H$, either $p_0 R(p_1, \ldots, p_{n-1})p_n \notin \mathcal{N}$ for all actual terms $p_n \in \mathcal{T}(\Sigma_0 \oplus \Sigma_1)$ (if R is a relation), or $p_0 R(p_1, \ldots, p_{n-1}) \notin \mathcal{N}$ (if R is a predicate). Since H involves only actual terms from $\mathcal{T}(\Sigma_0)$, in particular $p_0 \in \mathcal{T}(\Sigma_0)$. Thus, by definition of $\mathcal{N}$, either $p_0 R(p_1, \ldots, p_{n-1})p_n \notin \mathcal{M}$ for all $p_n \in \mathcal{T}(\Sigma_0 \oplus \Sigma_1)$, or $p_0 R(p_1, \ldots, p_{n-1}) \notin \mathcal{M}$, respectively. Hence, we may

conclude that $\mathcal{M} \models H$. Since $\mathcal{M}$ is a closed stable model for $T_0 \oplus T_1$, and there is a closed proof from $T_0 \oplus T_1$ for H/c, this implies $c \in \mathcal{M}$.

Since c contains only actual terms from $\mathcal{T}(\Sigma_0)$, in particular its left-hand side is in $\mathcal{T}(\Sigma_0)$, and so $c \in \mathcal{N}$.

2. Fix a transition $p_0 R(p_1, \ldots, p_{n-1}) p_n$ in $\mathcal{N}$. We show that there is a closed proof from T_0 for an actual rule $H/p'_0 R(p'_1, \ldots, p'_{n-1}) p'_n$, where H consists of negative transitions and $\mathcal{N} \models H$ and p'_i is α-convertible to p_i for $i = 1, \ldots, n$. (Similarly it can be proved that for each transition $p_0 R(p_1, \ldots, p_{n-1})$ in $\mathcal{N}$ there is a closed proof from T_0 for an actual rule $H/p'_0 R(p'_1, \ldots, p'_{n-1})$, where H consists of negative transitions and $\mathcal{N} \models H$.)

   Since $\mathcal{N} \subseteq \mathcal{M}$, the transition $p_0 R(p_1, \ldots, p_{n-1}) p_n$ is also in $\mathcal{M}$, which is a stable model for $T_0 \oplus T_1$. So there exists a closed proof P from $T_0 \oplus T_1$ for a closed actual rule $H/p'_0 R(p'_1, \ldots, p'_{n-1}) p'_n$ where H consists of negative transitions, and $\mathcal{M} \models H$ and p'_i is α-convertible to p_i for $i = 1, \ldots, n$. Since $\mathcal{N} \subseteq \mathcal{M}$, $\mathcal{M} \models H$ implies $\mathcal{N} \models H$. Remains to prove that P is a proof from T_0, which we will do by ordinal induction A on the length of P.

   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 $t^*_0 R(t^*_1, \ldots, t^*_{n-1}) t^*_n$ together with a formal substitution $\sigma^* : \mathcal{V}^* \to \mathcal{T}(\Sigma_0 \oplus \Sigma_1)$, where $\sigma^*(t^*_0) = p'_0$.

   First, we show that $\sigma^*(x^*) \in \mathcal{T}(\Sigma_0)$ for all $x^* \in SV(\rho(r^*))$, by induction B on the source distance of x^* in $\rho(r^*)$ (see definition 3.22).

   (a) $sd(\rho(r^*), x^*) = 0$.

   This means that $x^* \in EV(t^*_0)$. Since $\sigma^*(t^*_0) = p'_0$ is in $\mathcal{T}(\Sigma_0)$, lemma 3.14 yields $\sigma^*(x^*) \in \mathcal{T}(\Sigma_0)$.

   (b) $sd(\rho(r^*), x^*) = k + 1$.

   By definition there is a hypothesis $s^*_0 S(s^*_1, \ldots, s^*_{m-1}) s^*_m$ or $s^*_0 S(s^*_1, \ldots, s^*_{m-1})$ of $\rho(r^*)$ such that $x^* \in EV(s^*_i)$ for some $i = 1, \ldots, m$ and $sd(\rho(r^*), y^*) \leq k$ for all $y^* \in FV(s^*_0)$. Induction B implies that $\sigma^*(y^*) \in \mathcal{T}(\Sigma_0)$ for all $y^* \in FV(s^*_0)$. Furthermore, definition 3.19 of $\rho(r^*)$ ensures that $s^*_0 \in \mathbb{F}(\Sigma_0)$, so lemma 3.10 yields $\sigma^*(s^*_0) \in \mathcal{T}(\Sigma_0)$. The transition $\sigma^*(s^*_0 S(s^*_1, \ldots, s^*_{m-1}) s^*_m)$ or $\sigma^*(s^*_0 S(s^*_1, \ldots, s^*_{m-1})$ is proved by a strict sub-proof of P, so then ordinal induction A implies that T_0 proves this transition. In particular, $\sigma^*(s^*_i) \in \mathcal{T}(\Sigma_0)$ for $i = 1, \ldots, m$. Since $x^* \in EV(s^*_i)$ for some $i = 1, \ldots, m$, lemma 3.14 yields $\sigma^*(x^*) \in \mathcal{T}(\Sigma_0)$.

   Next, we show that r is in T_0. Suppose not, so let $r^* \in T_1$; we deduce a contradiction. Since $\sigma^*(t^*_0) = p'_0$ is in $\mathcal{T}(\Sigma_0)$, lemma 3.6 implies that t^*_0 is not new. Then by assumption there is a hypothesis in r^* of the form $s^*_0 S(s^*_1, \ldots, s^*_{m-1}) s^*_m$ or $s^*_0 S(s^*_1, \ldots, s^*_{m-1})$, where either S or some s^*_i for $i = 1, \ldots, m$ is new, and $s^*_0 \in \mathbb{F}(\Sigma_0)$, and $FV(s^*_0) \subseteq SV(\rho(r^*))$.

If s_i^* is new for some $i = 1, \ldots, m$, then lemma 3.6 says that $\sigma^*(s_i^*) \notin \mathbb{T}(\Sigma_0)$. Hence, since either S is new or $\sigma^*(s_i^*) \notin \mathbb{T}(\Sigma_0)$ for some $i = 1, \ldots, m$, the sub-proof of P of $H/\sigma^*(s_0^* S(s_1^*, \ldots, s_{m-1}^*)s_m^*)$ or $H/\sigma^*(s_0^* S(s_1^*, \ldots, s_{m-1}^*))$ cannot be a proof from T_0. So according to ordinal induction A, $\sigma^*(s_0^*) \notin \mathcal{T}(\Sigma_0)$. Since $s_0^* \in \mathbb{F}(\Sigma_0)$, lemma 3.10 yields $\sigma^*(x^*) \notin \mathcal{T}(\Sigma_0)$ for some $x^* \in FV(s_0^*) \subseteq SV(\rho(r^*))$. Contradiction.

So apparently r^* is in T_0. Then $\rho(r^*) = r^*$ (see Section 3.6), so $\sigma^*(x^*) \in \mathcal{T}(\Sigma_0)$ for all $x^* \in SV(r^*)$. By assumption T_0 is source-dependent, so the formal variables in r^* are all in $SV(r^*)$. Hence, $\sigma^*(x^*) \in \mathcal{T}(\Sigma_0)$ for all x^* in r^*.

Thus, $\sigma^*(r^*)$ involves only actual terms in $\mathcal{T}(\Sigma_0)$. 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 $H/\sigma^*(h^*)$ 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. □

Under the conditions from theorem 3.21, $T_0 \oplus T_1$ is not only a closed conservative extension, but also an open conservative extension of T_0. We formulate this in the next corollary.

Corollary 3.23 *Under the conditions from theorem 3.21, $T_0 \oplus T_1$ is an open conservative extension of T_0.*

Proof sketch. Consider free occurrences of actual variables in actual terms as constants. Then open actual terms and proofs become 'closed', so that the conservativity result for closed stable models applies. □

4 Applications

In this section we will give the reader an idea of the range of applications of our results. For a start, we wish to mention that Baeten and Verhoef [5] and Aceto, Bloom and Vaandrager [1] give several applications of our conservativity result in the case of operational semantics without types and binding mechanisms. Some systems with variable binding mechanisms to which our results can be applied are CCS from Milner [28], CSP from Hoare [23], the π-calculus from Milner, Parrow and Walker [30], and ACP with real-time as proposed by Fokkink and Klusener [17], to mention a few.

First, we will focus on a process algebra application with both types and variable binding: the π-calculus. Then we will devote our attention to the formulation of our results in conditional term rewriting; here also we will give examples.

4.1 An application in the π-calculus

We show how our conservativity results can be applied to a TSS from the literature that incorporates both types and variable binding. We opt for the πI-calculus from Sangiorgi [36], which is a subset of the full π-calculus. Basically, one could say that the πI-calculus is made out of CCS combined with α-conversion. The transition rules for the π-calculus as defined in [29] satisfy our criteria too, so our conservativity results can be applied to this formalism just as well. However, we prefer πI over π here, because it

has a simpler operational semantics, and we want to keep this exposition as smooth as possible.

We already encountered the πI-calculus, and its transition rule PRE, briefly in example 2.14. We will now explain its syntax and semantics in more detail. Recall that there are two sorts *Port* and *Process*. Process terms are defined by the following BNF grammar:

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

where 0 and t are terms of sort *Process*, and x and y are actual variables of sort *Port*. As usual, $t + t'$ denotes the alternative composition an $t|t'$ the communication merge. The process $x(y).t$ sends, and the process $\bar{x}(y).t$ reads, port name y via port x and proceeds as t. In both expressions, the x is free, and the y is bound in t. Finally, $\nu\ x\ t$ expresses that the port name x is made local in t, that is, the x is bound in t.

PRE	$x(y).v^* \xrightarrow{x(y)} v^*$	SUM	$\dfrac{v^* \xrightarrow{x(y)} v'^*}{v^* + w^* \xrightarrow{x(y)} v'^*}$
PAR	$\dfrac{v^* \xrightarrow{x(y)} v'^* \quad y \text{ not free in } w^*}{v^*\|w^* \xrightarrow{x(y)} v'^*\|w^*}$	COM	$\dfrac{v^* \xrightarrow{x(y)} v'^* \quad w^* \xrightarrow{\bar{x}(y)} w'^*}{v^*\|w^* \xrightarrow{\tau} \nu\ y\ (v'^*\|w'^*)}$
RES	$\dfrac{v^* \xrightarrow{x(y)} v'^*}{\nu\ z\ v^* \xrightarrow{x(y)} \nu\ z\ v'^*} \quad z \notin \{x, y\}$		

Table 1: Operational semantics of the πI-calculus

The operational semantics of the πI-calculus is presented in Table 1, 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 1 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.

We already encountered the rule PRE in example 2.14. Recall from that example that the y is a free parameter in the labels, but that it is a binder of v^* in the source of PRE.

The variables in the rules in Table 1 are all source-dependent. As an example, we show that this is the case for the rule COM. This rule 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 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 the right-hand sides v'^* and w'^* are source-dependent, respectively.

The side condition 'y not free in w^*' in PAR can be encoded as a predicate in a collection of source-dependent transition rules. This easy exercise is left to the reader. (See Baeten and Verhoef [4] and Verhoef [40] for many examples how to encode side conditions as predicates.) Hence, the operational semantics for the πI-calculus can be represented by a source-dependent TSS T_0.

Theorem 3.21 and corollary 3.23 imply that a well-defined sum $T_0 \oplus T_1$ is both a closed and an open conservative extension of the TSS T_0 for the πI-calculus if each rule r^* in T_1 satisfies one of two syntactic criteria:

- either the source of r^* is new,

- or one of the hypotheses of r^* has a special form, see theorem 3.21.

These two criteria cover types of extensions that are common in the literature. The first criterion can deal with the case that a rule in the extension T_1 describes the behaviour of a new function symbol, because in general this new function symbol will be present in the source of such a rule. The second criterion is often fulfilled if a function symbol in the original signature adopts a new meaning when applied to new terms.

Remark 4.1 In the πI-calculus, port names are not processes, but data that are used to parametrize processes. Since we do not distinguish between processes and data, in our setting port names are considered to be processes too. This means that our conservativity results are slightly stronger than necessary, namely, that behaviour of both processes (interesting) and port names (not so interesting) is not influenced by transition rules which satisfy one of the two criteria above.

4.2 Application to conditional term rewriting

There is a strong link between the world of structured operational semantics and that of conditional term rewriting. Both fields deal with what can be viewed as 'term deduction systems'. Terms are built from a set of function symbols, and binary relations on terms (transitions versus rewrite steps) are defined by means of proof rules (transition rules versus conditional rewrite rules). Such a rule, together with the validity, or non-validity, of a number of relations between terms, may imply the validity of another relation between terms.

There is only one real distinction: in conditional rewriting, provability is closed under context, in other words, if $s \to t$ is provable, then $C[s] \to C[t]$ is provable. The set of transitions provable from a TSS does not have to satisfy this characteristic, so in general a TSS cannot be expressed as a conditional term rewriting system (CTRS). However, we will see that the reverse transposition is possible, that is, for each CTRS there is an equivalent TSS. Essentially, this transformation is obtained by adding so-called context rules for all function symbols.

Hence, theorems in structured operational semantics also apply to the field of conditional rewriting. This link was already noted, but not exploited, by Groote and Vaandrager [22]. They refrain from transposing their results to conditional rewriting because it would yield lemmas that do not serve any practical purpose.

We show in this section that our conservativity results do constitute useful lemmas in the field of conditional rewriting. They formulate sufficient syntactic requirements for CTRSs R_0 and R_1 to ensure that the rewrite relation induced by R_0 on original terms will not be affected if R_0 is extended with R_1. This result is applicable in the field of abstract data types, which are often specified by means of modules of CTRSs. In abstract data typing there is a long tradition in modular specifying systems. In fact, modular specifying means extending conservatively, in our terminology; see [7] for more information.

Our conservativity theorem serves only as an example how theorems from structured operational semantics can be transposed to conditional rewriting. In order to keep this exposition clean, we will not include important features from the previous sections, such as variable binding, substitution harnesses, formal variables, predicates, and negative conditions. These features are certainly relevant for the field of conditional rewriting. For instance, it is well-known that the λ-calculus can be viewed as a term rewriting system, which involves variable bindings and substitution harnesses. For example, the rule for β-reduction is:

$$(\lambda x.M)N \to M[x := N].$$

We can argue, analogously to the SOS case, that the substitution at the right-hand side is not a real substitution, but a substitution harness. Even so, negative conditions are relevant for conditional rewriting, and have been studied for instance by Kaplan [26]. In order to find out how the conservativity results apply to CTRSs with variable bindings and negative conditions, the reader is referred to the previous sections.

Remark 4.2 In structured operational semantics we allowed negative premises of the form $s \not\longrightarrow$ for relations $\longrightarrow$, see definition 2.10. This construct expresses that there does not exits a relation $s \longrightarrow t$ for any t, or in other words, $\forall x\ (s \not\longrightarrow x)$, where x is a fresh variable. In order to make these expressions suited for conditional rewriting, they are to be generalized to the form $\forall x_1, \ldots, x_n\ (s \not\longrightarrow t)$, where $x_1, \ldots, x_n$ are the variables that occur in t but not in s; see Van de Pol [34] for more detailed information. CTRSs with such negative conditions can be given a meaning following Van Glabbeek [20].

We define the necessary preliminaries concerning conditional term rewriting and discuss the connection with our results. We will provide some examples from the term rewriting literature to exemplify the results. For detailed information on conditional rewriting we refer to Kaplan [25] and Bergstra and Klop [9].

Bergstra and Klop [9] classify the various kinds of CTRSs that occur in the literature into four types: I (or 'semi-equational'), II (or 'join'), III and III$_n$ (or 'normal'). These types differ in the meaning that is given to the conditions that occur the rewrite rules. In this section, we will focus on CTRSs of type III. A CTRS of any of the other three types can always be transformed into a CTRS of type III. We will see that our conservativity results are useful for CTRSs of the types II and III and III$_n$. The transformation of

type I to type III yields CTRSs that are outside the scope of our conservativity results. We will leave it as an open problem to solve this matter.

Definition 4.3 *A* conditional term rewriting system *(CTRS) of type III is a pair* (Σ, R) *with* Σ *a (formal) signature and* R *a set of* conditional rewrite rules *of the form*

$$l \to r \Leftarrow s_1 \twoheadrightarrow t_1, \ldots, s_n \twoheadrightarrow t_n,$$

where l, r, s_i, t_i *are terms over the signature* Σ*. The expressions* $s_i \twoheadrightarrow t_i$ *are called the* conditions *of the rewrite rule, and* $\twoheadrightarrow$ *denotes the transitive-reflexive closure of the one-step rewrite relation* $\to$*.*

A rewrite rule without conditions will be written as $l \to r$, provided that no confusion can arise.

Remark 4.4 On a conditional rewrite rule we often see the the following three conditions.

A. The left-hand side l is not a variable.

B. The variables that occur in the right-hand side r also occur in the left-hand side l.

C. No extra variables occur in the conditions.

Restrictions A and B are quite natural in the unconditional case, because then they are essential in order to obtain termination. According to Middeldorp [27], restriction C is often imposed to prevent severe complications of a technical nature.

We leave out these restrictions, because our results do not require to impose them. However, we will see that for CTRSs of types II and III_n, the conditions of the conservativity theorem translate to the restrictions B and C.

In the previous sections, on structured operational semantics, we allowed negative premises in the transition rules. In order to give meaning to TSSs, we introduced the notion of a 'stable model', from Gelfond and Lifschitz [18], see definition 2.20. Since in this part on conditional rewriting we have abstracted from negative conditions, it is much easier to give meaning to CTRSs. Recall that $\twoheadrightarrow$ denotes the transitive-reflexive closure of $\to$.

Definition 4.5 $s \to t$ *is provable from a CTRS if there exists a rule* $l \to r \Leftarrow s_1 \twoheadrightarrow t_1, \ldots, s_n \twoheadrightarrow t_n$ *in the CTRS, a substitution* σ*, and a context* $C[\ ]$ *such that* $s \equiv C[\sigma(l)]$ *and* $t \equiv C[\sigma(r)]$ *and* $\sigma(s_i) \twoheadrightarrow \sigma(t_i)$ *is provable from the CTRS for* $i = 1, \ldots, n$*.*

Before we continue, we give an example of a CTRS, taken from [15], which will serve as a running example to demonstrate our conservativity result.

Example 4.6 The following CTRS of type III is built from two modules N_0 and N_1. The CTRS N_0 implements addition on natural numbers. It assumes the sort $\mathbb{N}$ of natural

numbers, together with the constant 0, the successor function $S : \mathbb{N} \to \mathbb{N}$, and addition $A : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$. The CTRS N_0 consists of the following two rules:

$$\begin{cases} A(0, x) \to x \\ A(S(x), y) \to S(A(x, y)) \end{cases}$$

The CTRS N_1 implements the Fibonacci numbers. It assumes the sorts $\mathbb{N}$ and $\mathbb{N} \times \mathbb{N}$, the constant 0, and the functions S and A and $Fib : \mathbb{N} \to \mathbb{N} \times \mathbb{N}$. The CTRS N_1 consists of the following two rules:

$$\begin{cases} Fib(0) \to (0, S(0)) \\ Fib(S(x)) \to (z, A(y, z)) \Leftarrow Fib(x) \twoheadrightarrow (y, z) \end{cases}$$

The CTRS N_1 is considered difficult, because the variables y and z do not occur in the left-hand side of the rule. Later on we will see that our conservativity format applies to $N_0 \oplus N_1$ without any problem.

Our conservativity results make heavy use of the notion of *source-dependency*, which was introduced for TSSs in definition 3.16. We transpose this definition to CTRSs of type III.

Definition 4.7 *Assume a conditional rewrite rule of type III:*

$$l \to r \Leftarrow s_1 \twoheadrightarrow t_1, \ldots, s_n \twoheadrightarrow t_n.$$

We define inductively what are the source-dependent variables of this rule.

- *The variables in l are source-dependent.*
- *If the variables in s_i are all source-dependent, then the variables in t_i are source-dependent.*

We say that a conditional rewrite rule of type III is source-dependent if all the variables in the rule are source-dependent. A CTRS of type III is source-dependent if all its rules are so.

Example 4.8 We show that the CTRS $N_0 \oplus N_1$ in example 4.6 is source-dependent. The first three rules are trivial because they do not have conditions, and they satisfy restriction B in remark 4.4: the variables that occur in the right-hand side also occur in the left-hand side. The fourth rule incorporates three variables: x, y, and z. Since $l = Fib(S(x))$, it follows that x is source-dependent. Since x is the only variable in the left-hand side $Fib(x)$ of the condition, the variables in the right-hand side (y, z) are also source-dependent. So the fourth rule is source-dependent too.

We briefly discuss the three types I, II and III_n for CTRSs from [9], and how they can be transformed to type III CTRSs. We will find that if a CTRS of type II or III_n satisfies requirements B and C of remark 4.4, then its transformation to type III is source-dependent.

- In type III_n, conditions are conjuncts of expressions $s \twoheadrightarrow t$ where t is a ground normal form. In particular, terms at the right-hand sides of conditions are closed, so it follows that the only source-dependent variables in type III_n rules are the ones in the left-hand side of the rule. Hence, a rule of type III_n is source-dependent if it satisfies requirements B and C of remark 4.4, which together ensure that all variables in the rule occur in the left-hand side.

- In type II, conditions are conjuncts of expressions $s \downarrow t$, which denote that both s and t reduce to a term u. This can be formulated in type III style: $s \twoheadrightarrow y$ and $t \twoheadrightarrow y$ where y is a fresh variable. Since the y is fresh, by definition it does not occur in any of the conditions. So the only source-dependent variables in type II rules are the ones in the left-hand side of the rule. Hence, a rule of type II is source-dependent if it satisfies requirements B and C of remark 4.4.

- Finally, in type I, conditions are conjuncts of expressions $s = t$, which denote that s rewrites to t if the rewrite rules may be applied both from left to right and from right to left. There is a laborious way to express a rule of type I in infinitely many rules of type III. However, the rules that are thus obtained are not source-dependent, so that the conservativity results do not apply. We leave it as an open problem to solve this matter, if possible at all.

4.3 The conservativity theorems in conditional rewriting

In this subsection we discuss the conservativity theorems in the case of CTRSs.

Definition 4.9 *Let (Σ_0, R_0) and (Σ_1, R_1) be two CTRSs of type III. $R_0 \oplus R_1$ is a* closed/open conservative extension *of R_0 if for every closed/open term s over Σ_0, s rewrites to a term t in the original system R_0 if and only if s rewrites to t in the extended system $R_0 \oplus R_1$.*

In theorem 3.21 and corollary 3.23 we formulated under which conditions an extension of a TSS is both open and closed conservative. Here, we transpose these results to the setting of conditional rewriting. One of the clauses in theorem 3.21 leaves the possibility for a rule in the extension to have an original term at the left-hand side. We leave out this clause here, in order to keep this exposition simple.

Theorem 4.10 *Assume two type III CTRSs (Σ_0, R_0) and (Σ_1, R_1). If R_0 is source-dependent, and if in each rule in R_1, the term at the left-hand side contains a function symbol from $\Sigma_1 \backslash \Sigma_0$, then $R_0 \oplus R_1$ is both a closed and an open conservative extension of R_0.*

Proof. We show how a type III CTRSs can be translated into an equivalent TSS. Then we transform R_0 and $R_0 \oplus R_1$ into equivalent TSSs, and show that the open and closed conservativity theorems for TSSs, theorem 3.21 and corollary 3.23, apply.

Assume a CTRS (Σ, R) of type III. We construct a TSS T such that R proves a rewrite step $s \to t$ if and only T proves the transition $s \to t$. Each conditional rule

$l \to r \Leftarrow s_1 \twoheadrightarrow t_1, \ldots, s_n \twoheadrightarrow t_n$ in R can be expressed as a transition rule in T:

$$\frac{\{s_1 \twoheadrightarrow t_1, \ldots, s_n \twoheadrightarrow t_n\}}{l \to r}.$$

If the notions of provability in a CTRS and a TSS would coincide, we would now be ready with the translation. However, there is one essential distinction: in conditional rewriting, provability is closed under context, that is, if $s \to t$ is provable then $C[s] \to C[t]$ is provable. In order to bridge this gap, we add context rules to T for all function symbols $f \in \Sigma$:

$$\frac{x_i \to y}{f(x_1, \ldots, x_i, \ldots, x_n) \to f(x_1, \ldots, y, \ldots, x_n)}, \qquad i = 1, \ldots, n.$$

These context rules are source-dependent. Namely, the variables $x_1, \ldots, x_n$ occur in the source, so they are source-dependent, and the condition $x \to y$ ensures that y is source-dependent.

Finally, in order to define the transitive-reflexive closure $\twoheadrightarrow$ of $\to$, we need two more transition rules in T:

$$x \twoheadrightarrow x \qquad \text{and} \qquad \frac{x \to y \quad y \twoheadrightarrow z}{x \twoheadrightarrow z}$$

Again, these transition rules are source-dependent. Note that they do not satisfy requirement A in remark 4.4 i.e., their sources are single variables.

It is not hard to see that R proves a rewrite step $s \to t$ if and only if T proves the transition $s \to t$.

We apply the strategy described above in order to translate R_0 and $R_0 \oplus R_1$ into equivalent TSSs T_0 and $T_0 \oplus T_1$, respectively.

1. The TSS T_0 consists of the transformations of the conditional rules in R_0 into transition rules, together with the context rules for function symbols in Σ_0, and the two transition rules for $\twoheadrightarrow$.

2. The TSS T_1 contains the transformations of the conditional rules in R_1 into transition rules, together with the context rules for function symbols in $\Sigma_1 \backslash \Sigma_0$.

We verify that T_0 and T_1 meet the requirements from theorem 3.21.

1. Since the conditional rules in R_0 are source-dependent, it follows that their transformations into transition rules are source-dependent. Moreover, the context rules and the two transition rules for $\twoheadrightarrow$ are source-dependent. Hence, the TSS T_0 is source-dependent.

2. Since the conditional rules in R_1 contain a function symbols from $\Sigma_1 \backslash \Sigma_0$ in their left-hand side, the same holds for their transformations into transition rules. Moreover, the context rules for function symbols in $\Sigma_1 \backslash \Sigma_0$ contain a function symbols from $\Sigma_1 \backslash \Sigma_0$ in their source.

So according to theorem 3.21 and corollary 3.23, $T_0 \oplus T_1$ is both a closed and an open conservative extension of T_0. Since R_0 or $R_0 \oplus R_1$ are equivalent with T_0 and $T_0 \oplus T_1$ respectively, it follows that $R_0 \oplus R_1$ is both a closed and an open conservative extension of R_0. $\Box$

Example 4.11 We show that the conservativity theorem 4.10 applies to the CTRS $N_0 \oplus N_1$ from example 4.6. Recall from example 4.8 that N_0 and N_1 are source-dependent.

Since N_0 is source-dependent, and since the left-hand sides of the two rules in N_1 both contain the function symbol *Fib*, theorem 4.10 yields that $N_0 \oplus N_1$ is a closed and an open conservative extension of N_0.

Furthermore, $N_0 \oplus N_1$ is source-dependent, so we can extend this CTRS in a conservative way, as long as we make sure that the rules in the extension have a new function symbol in their left-hand sides. For example, let the CTRS N_2 implement multiplication $M : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ as follows.

$$\begin{cases} M(0, x) \to 0 \\ M(S(x), y) \to A(M(x, y), y) \end{cases}$$

Since the new function symbol M occurs in the left-hand sides of the two rules of N_2, it follows that $N_0 \oplus N_1 \oplus N_2$ is a conservative extension of $N_0 \oplus N_1$.

5 Conclusions

In this paper we set up a formal framework to describe term deduction systems such as appear in structured operational semantics in the style of Plotkin and in conditional term rewriting. This framework has the power to express many-sortedness, general binding mechanisms and substitutions, among other notions such as negative premises and unary predicates on terms. This framework can serve as a platform to prove general statements concerning such systems.

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

Furthermore, we presented a general strategy to translate theorems in structured operational semantics to conditional term rewriting systems. In particular, we have applied this approach to the conservativity theorem, which yields a result that is useful for abstract data terms, when specified as modules of conditional term rewriting systems.

We showed that these conservativity results are applicable to many and diverse process algebras and conditional term rewriting systems. For example, we have seen that they can be applied successfully in the setting of the π-calculus.

References

[1] L. Aceto, B. Bloom, and F.W. Vaandrager. Turning SOS rules into equations. *Information and Computation*, 111(1):1–52, 1994.

[2] S.F. Allen, R.L. Constable, D.J. Howe, and W.E. Aitken. The semantics of reflected proof. In *Proceedings LICS'90*, pp. 95–197. IEEE Computer Society Press, 1990.

[3] J.C.M. Baeten and J.A. Bergstra. Discrete time process algebra. In *Proceedings CONCUR'92*, LNCS 630, pp. 401–420. Springer, 1992. Revised version to appear in *Formal Aspects of Computing*.

[4] J.C.M. Baeten and C. Verhoef. A congruence theorem for structured operational semantics with predicates. In *Proceedings CONCUR'93*, LNCS 715, pp. 477–492. Springer, 1993.

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

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

[7] J.A. Bergstra, J. Heering, and P. Klint, editors. *Algebraic Specification*. ACM Press in cooperation with Addison Wesley, 1989.

[8] J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. *Information and Control*, 60(1/3):109–137, 1984.

[9] J.A. Bergstra and J.W. Klop. Conditional rewrite rules: confluence and termination. *Journal of Computer and System Sciences*, 32:323–362, 1986.

[10] B. Bloom. Structural operational semantics for weak bisimulations. *Theoretical Computer Science*, 146(1/2):25–68, 1995.

[11] B. Bloom, S. Istrail, and A.R. Meyer. Bisimulation can't be traced. *Journal of the ACM*, 42(1):232–268, 1995.

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

[13] R.N. Bol and J.F. Groote. The meaning of negative premises in transition system specifications. In *Proceedings ICALP'91*, LNCS 510, pp. 481–494. Springer, 1991.

[14] R.L. Constable, S.F. Allen, H.M. Bromley, W.R. Cleaveland, J.F. Cremer, R.W. Harper, D.J. Howe, T.B. Knoblock, N.P. Mendler, P. Panangaden, J.T. Sasaki, and S.F. Smith. *Implementing Mathematics with the Nuprl Proof Development System*. Prentice Hall International, 1986.

[15] N. Dershowitz, M. Okada, and G. Shivkumar. Confluence of conditional rewrite systems. In *Proceedings CTRS'87*, LNCS 308, pp. 31–44. Springer, 1987.

[16] W.J. Fokkink and R.J. van Glabbeek. Ntyft/ntyxt rules reduce to ntree rules. Technical Note CS-95-17, Stanford University, 1995. To appear in *Information and Computation*.

[17] W.J. Fokkink and A.S. Klusener. An effective axiomatization for real time ACP. Report CS-R9542, CWI, Amsterdam, 1995. To appear in *Information and Computation*.

[18] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proceedings 5th Conference on Logic Programming*, pp. 1070–1080. MIT press, 1988.

[19] R.J. van Glabbeek. Full abstraction in structural operational semantics. In *Proceedings AMAST'93*, Workshops in Computing, pp. 77–84. Springer, 1993.

[20] R.J. van Glabbeek. The meaning of negative premises in transition system specifications II. Technical Note CS-95-16, Stanford University, 1995.

[21] J.F. Groote. Transition system specifications with negative premises. *Theoretical Computer Science*, 118(2):263–299, 1993.

[22] J.F. Groote and F.W. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Information and Computation*, 100(2):202–260, 1992.

[23] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice Hall International, 1985.

[24] D.J. Howe. Proving congruence of bisimulation in functional programming languages. Technical Report, AT&T Bell Laboratories, 1995.

[25] S. Kaplan. Conditional rewrite rules. *Theoretical Computer Science*, 33(2):175–193, 1984.

[26] S. Kaplan. Positive/negative conditional rewriting. In *Proceedings CTRS'87*, LNCS 308, pp. 129–143. Springer, 1987.

[27] A. Middeldorp. Modular properties of conditional term rewriting systems. *Information and Computation*, 104(1):110–158, 1993.

[28] R. Milner. *Communication and Concurrency.* Prentice Hall International, 1989.

[29] R. Milner, J. Parrow, and D. Walker. Modal logics for mobile processes. In *Proceedings CONCUR'91*, LNCS 527, pp. 45–60. Springer, 1991.

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

[31] F. Moller and C. Tofts. A temporal calculus of communicating systems. In *Proceedings CONCUR'90*, LNCS 458, pp. 401–415. Springer, 1990.

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

[33] G.D. Plotkin. A structural approach to operational semantics. Report DAIMI FN-19, Aarhus University, 1981.

[34] J.C. van de Pol. Semantics of priority rewrite systems by means of transition system specifications. Unpublished manuscript, 1995.

[35] D. Sangiorgi. The lazy lambda calculus in a concurrency scenario. *Information and Computation*, 111(1):120–153, 1994.

[36] D. Sangiorgi. πI: A symmetric calculus based on internal mobility. In *Proceedings TAPSOFT'95*, LNCS 915, pp. 172–186. Springer, 1995.

[37] R. de Simone. Higher-level synchronising devices in MEIJE–SCCS. *Theoretical Computer Science*, 37:245–267, 1985.

[38] A. Stoughton. Substitution revisited. *Theoretical Computer Science*, 59:317–325, 1988.

[39] C. Verhoef. A general conservative extension theorem in process algebra. In *Proceedings PROCOMET'94*, *IFIP Transactions* A-56, pp. 149–168. Elsevier, 1994.

[40] C. Verhoef. A congruence theorem for structured operational semantics with predicates and negative premises. *Nordic Journal of Computing*, 2(2):274–302, 1995.

[41] D.A. Watt. *Programming Concepts and Paradigms.* Prentice Hall International, 1990.