Variable binding operators in transition system specifications

Citation for published version (APA):

Document status and date:
Published: 01/01/2000

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher’s website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.
Variable Binding Operators
in Transition System Specifications

by

C.A. Middelburg

00/07

ISSN 0926-4515

All rights reserved
editors: prof.dr. J.C.M. Baeten
prof.dr. P.A.J. Hilbers

Reports are available at:
http://www.win.tue.nl/win/cs

Computing Science Reports 00/07
Eindhoven, March 2000
Variable Binding Operators
in Transition System Specifications

C.A. Middelburg

Computing Science Department, Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, the Netherlands

Department of Philosophy, Utrecht University
P.O. Box 80126, 3508 TC Utrecht, the Netherlands

mailto:keesm@win.tue.nl, http://www.win.tue.nl/~keesm

Abstract

In this paper the approach to structural operational semantics (SOS) using transition system specifications (TSSs) is extended to deal with variable binding operators and many-sortedness. Verhoef's transition rule format, known as the panth format, generalizes naturally to the new TSSs. It is shown that, in case of a meaningful TSS defining ordinary transition relations, bisimulation is a congruence if transition rules in this format are used. Formats guaranteeing that bisimulation is a congruence are important for the application of TSSs to provide process calculi, and programming and specification languages, with an operational semantics. The new congruence result is relevant because in many of these applications, variable binding operators and many-sortedness are involved. It is also sketched how the presented approach can be further extended to deal with given sorts and parametrized transition relations. Given sorts are useful if the semantics of the terms of certain sorts has been given beforehand. This happens frequently in practice, as does the often related need of parametrized transition relations.

Keywords & Phrases: structural operational semantics (SOS), transition system specifications (TSSs), panth format, bisimulation, congruence, many-sortedness, variable binding operators, binding algebras, partial stable models.


1 Introduction

In [20], an approach to structural operational semantics using transition system specifications (TSSs) was introduced. The approach considers transition systems where the states are the closed terms over a given signature. The original TSSs define binary transition relations by means of transition rules with positive premises. The approach has first been extended in [18, 7] to transition rules with positive and negative premises and next in [31] to the specification of unary and binary relations. In all these cases, variable binding operators and many-sortedness are not supported.
In many applications of TSSs, it is convenient to use negative premises or to define unary relations (see e.g. [18] and [31]). Negative premises can often be avoided, but at the expense of simplicity. Representation of unary relations by binary relations is possible, but this trick does not contribute to comprehensibility.

In many applications of TSSs, it is in addition necessary to have support for many-sortedness or variable binding operators. Many-sortedness is found, for example, in process algebras with timing (see e.g. [4], [8] and [25]). Examples of variable binding operators are the integration operator ∫ of real time ACP [4] and the recursion operator μ of CSP [21] and CCS [23]. Using transition rules to cope with many-sortedness is unpractical and obscures the fact that it is a static matter. Variable binding operators can not be coped with at all without further extension. In [14], an extension to deal with variable binding operators and many-sortedness is proposed. Another one is proposed in this paper. An important difference between the extension presented in [14] and the one presented in this paper is that in the latter distinction between formal and actual variables, formal and actual terms, formal and actual substitutions, etc. is not made. This leads to TSSs that are more closely related to the original ones than the TSSs of [14]. As a consequence, the transition rule format introduced in [31], known as the panth format, generalizes naturally to the new TSSs.

The main difference with the original TSSs is that terms are used in which operators may bind variables in their arguments and variables may have arguments. The terms concerned are essentially the binding terms investigated in [30]. Similar expressions were also part of the meta-language used in [2] to introduce Frege structures. Variables that may have arguments are also known from combinatory reduction systems [22], where they occur as meta-variables in schematic rewrite rules for term rewriting with bound variables. Having variables that may have arguments obviates the need to distinguish two kinds of variables, terms, substitutions, etc. in TSSs. Such a distinction, which is made in [14], hinders generalization of definitions and results concerning TSSs without support for variable binding operators. Besides, it is doubtful that the complexity introduced by the distinction pays off in terms of the extent of applicability.

The meaning of TSSs proposed in [7], and reformulated in [17], also generalizes naturally to the TSSs presented in this paper. In [7], the meaning of a TSS is defined in a way that facilitates proving certain theorems related to the use of stratification (see e.g. [18]) as a technique to check if a TSS is meaningful. In [17], it is defined in a way that makes comparison with potential alternatives easier. In this paper, the meaning of TSSs is defined in still another way aimed at clarity in this intricate issue. It goes without saying that the different definitions agree with each other.

Support for many-sortedness has some interesting consequences. It happens frequently in practice that the semantics of the terms of certain sorts, called given sorts, has been given beforehand. The sort that represents the time domain in process algebras with timing is a typical example. It is impractical and unnecessary to redefine the semantics of the terms of given sorts. Furthermore, distinguishing given sorts makes it possible to relax the panth format and to deal with transition relations parametrized by terms of given sorts. This is also discussed in this paper.

The structure of this paper is as follows. First of all, in Section 2, we present binding terms and TSSs that define transition relations on binding terms. Binding algebras, the structures in which binding terms are interpreted, are presented in Sec-
tion 2 as well. In Section 3, we discuss the meaning of TSSs. Then, in Section 4, we first define the bisimulation equivalence induced by TSSs and the panth format for TSSs, and then show that for meaningful TSSs in panth format bisimulation equivalence is a congruence. After that, in Section 5, we explain how to deal with given sorts and parametrized transition relations using the TSSs introduced in Section 2. Finally, in Section 6, we make some concluding remarks.

2 Binding terms and TSSs

In this section, we first introduce the notions of binding term and binding algebra. The latter are the structures in which binding terms are interpreted. Next, we generalize the notion of TSS from conventional terms to binding terms. The meaning of TSSs is discussed in Section 3.

Abstract notions of binding term and binding algebra were introduced in [11]. The kind of binding terms and binding algebras introduced in this section are essentially many-sorted versions of the ones that were first introduced in [30].

2.1 Binding terms

We define terms over a many-sorted signature, roughly speaking a collection of sorts and operators, and a variable domain. Therefore, we first define the notions of binding sort, many-sorted binding signature and variable domain.

We assume a set \( \mathcal{S} \) of base sorts. A base sort stands for a set of which the elements are called ordinary objects. A binding sort is either a base sort or a sort that stands for a set of functions from sets corresponding to base sorts to a set corresponding to a base sort. As explained below, sorts of the latter kind are used for variable binding in arguments of operators.

Definition 2.1 Let \( \mathcal{S} \subseteq \mathcal{S} \). Then the set \( \mathcal{B}(\mathcal{S}) \) of binding sorts over \( \mathcal{S} \) is inductively defined by the following formation rules \( (n > 0) \):

1. \( \mathcal{S} \subseteq \mathcal{B}(\mathcal{S}) \);
2. \( s_1, \ldots, s_n, s \in \mathcal{S} \Rightarrow s_1, \ldots, s_n \cdot s \in \mathcal{B}(\mathcal{S}) \).

We assume a \( \mathcal{B}(\mathcal{S})^* \times \mathcal{S} \)-indexed family of mutually disjoint sets of binding operators \( \mathcal{O} = \langle \mathcal{O}_o \rangle_{\mathcal{O} \in \mathcal{B}(\mathcal{S})^* \times \mathcal{S}} \). We also assume a \( \mathcal{B}(\mathcal{S}) \)-indexed family of mutually disjoint, countably infinite sets of variables \( \mathcal{V} = \langle \mathcal{V}_s \rangle_{s \in \mathcal{B}(\mathcal{S})} \). Furthermore, it is assumed that \( \bigcup_{\mathcal{O} \in \mathcal{B}(\mathcal{S})^* \times \mathcal{S}} \mathcal{O}_o \) and \( \bigcup_{s \in \mathcal{B}(\mathcal{S})} \mathcal{V}_s \) are disjoint. We write \( o : s_1 \times \ldots \times s_n \rightarrow s \) to indicate that \( o \in \mathcal{O}_{(s_1, \ldots, s_n), s} \) and we write \( x : s \) to indicate that \( x \in \mathcal{V}_s \). A variable \( x \) is called ordinary if \( x : s \) for some \( s \in \mathcal{S} \).

An operator \( o : s_1 \times \ldots \times s_n \rightarrow s \) has \( n \) arguments. If \( s_i = s_{i1}, \ldots, s_{im_i} \cdot s_i \) \( (1 \leq i \leq n) \), then it binds \( n_i \) variables, of base sorts \( s_{i1}, \ldots, s_{im_i} \), in the \( i \)-th argument. Otherwise, i.e. if \( s_i \in \mathcal{S} \), it does not bind any variable in the \( i \)-th argument.

Definition 2.2 A (many-sorted) binding signature is a pair \( \Sigma = (\mathcal{S}, \mathcal{O}) \), with \( \mathcal{S} \subseteq \mathcal{S} \) and \( \mathcal{O} \subseteq \mathcal{O} \), such that for all \( o \in \mathcal{O} \), if \( o : s_1 \times \ldots \times s_n \rightarrow s \), then \( s_1, \ldots, s_n, s \in \mathcal{B}(\mathcal{S}) \).
Definition 2.3 Let $\Sigma = (S, O)$ be a binding signature. Then a variable domain for signature $\Sigma$ is a set $X \subseteq \bigcup_{s \in B(S)} V_s$ such that for all $s \in S$, $V_s \subseteq X$. For every $s \in B(S)$, we write $X_s$ for the set $\{x \in X \mid x : s\}$. We write $X_\Sigma$ and $V_\Sigma$ for the variable domains $\bigcup_{s \in S} V_s$ and $\bigcup_{s \in B(S)} V_s$, respectively.

Next, in Definition 2.4, we define the notion of binding term. Formation rule 2 shows that variables, with the exception of ordinary variables, have arguments. Variables are bound in terms formed by formation rule 3. Notice that the terms formed by application of this rule serve only as arguments of operators.

Definition 2.4 Let $\Sigma = (S, O)$ be a binding signature and $X$ be a variable domain for $\Sigma$. Then $T_\Sigma(X) = (T_\Sigma(X)_s)_{s \in B(S)}$, the family of sets of binding terms over signature $\Sigma$ and variables $X$, is inductively defined by the following formation rules:

1. $s \in S$, $x : s \Rightarrow x \in T_\Sigma(X)_s$;
2. $s_1, \ldots, s_n \in B(S)$, $x : s_1, \ldots, s_n : s$, $t_1 \in T_\Sigma(X)_{s_1}, \ldots, t_n \in T_\Sigma(X)_{s_n} \Rightarrow x(t_1, \ldots, t_n) \in T_\Sigma(X)_s$;
3. $s_1, \ldots, s_n : s \in B(S)$, $x_1 : s_1, \ldots, x_n : s_n, x_1, \ldots, x_n$ mutually distinct, $t \in T_E(X)_s \Rightarrow x_1, \ldots, x_n : t \in T_E(X)_{s_1, \ldots, s_n, s}$;
4. $o \in O$, $o : s_1 \times \ldots \times s_n \Rightarrow s$, $t_1 \in T_E(X)_{s_1}, \ldots, t_n \in T_E(X)_{s_n} \Rightarrow o(t_1, \ldots, t_n) \in T_E(X)_s$.

In $o(t_1, \ldots, t_n)$, we usually omit the parentheses whenever $n = 0$. We write $T_\Sigma$ for $T_\Sigma(V_\Sigma)$. For $t \in \bigcup_{s \in B(S)} T_\Sigma(X)_s$, we write $s(t)$ for the $s \in B(S)$ such that $t \in T_\Sigma(X)_s$.

Example 2.5 In CCS [23], the operator $\mu$ is used to define processes recursively. For example, the expression $\mu x . ax$ denotes the solution of the equation $x = ax$, i.e. the process that will keep on performing action $a$ forever. The recursion operator $\mu$ is actually a unary variable binding operator that binds one variable in its argument. The expression $\mu x . ax$ abbreviates the binding term $\mu(x . ax)$.

The following definition makes the notion of closed binding term precise. It also introduces the notions of free and bound occurrence of a variable.

Definition 2.6 An occurrence of a variable $x$ in a term $t$ is bound if the occurrence is in the term $t'$ of a subterm of the form $x_1, \ldots, x_n : t'$ where $x \in \{x_1, \ldots, x_n\}$; otherwise it is free. If $x$ has at least one free occurrence in $t$, it is called a free variable of $t$. A term $t$ is closed if it is a term without free variables. We write $C T_\Sigma(X)$ for the family of sets of closed binding terms $(\{t \in T_\Sigma(X)_s \mid t \text{ is closed}\})_{s \in B(S)}$. We write $C T_\Sigma$ for $C T_\Sigma(V_\Sigma)$.

Substitution of binding terms for variables is needed in many occasions. We first define a notion of substitution restricted to ordinary variables. It allows us to define the notion of term algebra in the setting of binding algebras (see Section 2.2). Using the notion of term algebra, we will define a notion of substitution that is not restricted to ordinary variables. Notice that free and bound occurrences of variables are treated differently in substitution.
Definition 2.7 Let $\Sigma = (S, O)$ be a binding signature and $X$ be a variable domain for $\Sigma$. Then an ordinary substitution $\sigma : X \to T_E(X)$ of terms in $T_E(X)$ for ordinary variables in $X$ is an $S$-indexed family of functions $(\sigma_s : X_s \to T_E(Y)_s)_{s \in S}$. An ordinary substitution $\sigma$ extends from ordinary variables to terms in the usual way: $\sigma_s(t)$ is the term obtained by simultaneously replacing in $t$ all free occurrences of ordinary variables $x$ by $\sigma_s(x)(x)$, renaming bound occurrences of ordinary variables in $t$ if needed to avoid free occurrences of variables in the replacing terms becoming bound. For every ordinary substitution $\sigma : X \to T_E(X)$ and $t \in T_E(X)_s$, we write $\sigma(t)$ or $\sigma(t)$ for $\sigma_s(t)$. We write $[t_1, \ldots, t_n/x_1, \ldots, x_n]$ for the ordinary substitution $\sigma$ such that $\sigma(x_1) = t_1, \ldots, \sigma(x_n) = t_n$ and $\sigma(x) = x$ if $x \notin \{x_1, \ldots, x_n\}$.

Notice that ordinary substitution is defined up to change of bound variables. This does not pose any problem, because binding terms that can be obtained from each other by change of bound variables are semantically equivalent (see Definition 2.9). Indeed, we will introduce an equivalence relation formalizing this identification (see Definition 2.10).

2.2 Binding algebras

Binding terms are interpreted in binding algebras. Binding algebras constitute a restricted kind of second-order algebras, suitable to deal with variable binding operators, which can be regarded as an algebraic generalization of the Frege structures introduced in [2].

We define binding algebras with respect to a binding signature. An important condition to be satisfied by a binding algebra is that each term can be given an interpretation in it for any object or function its free variables may stand for. Obviously, the interpretation of a term depends on the objects and functions that are associated with its free variables. Assignments, which are defined first, model such associations. They can be viewed as semantic counterparts of substitutions that are not restricted to ordinary variables (see Definition 2.13).

Definition 2.8 Let $\Sigma = (S, O)$ be a binding signature and $X$ be a variable domain for $\Sigma$. Furthermore, let $D$ be a $B(S)$-indexed family of sets $(D_s)_{s \in B(S)}$. Then an assignment $\alpha : X \to D$ of values in $D$ to variables in $X$ is a $B(S)$-indexed family of functions $(\alpha_s : X_s \to D_s)_{s \in B(S)}$. We write $[X \to D]$ for the set of all assignments $\alpha : X \to D$. For every assignment $\alpha : X \to D$ and $x : s$, we write $\alpha(x)$ for $\alpha_s(x)$. For every assignment $\alpha : X \to D$, $x : s$ and $d \in D_s$, we write $\alpha(x \to d)$ for the $\alpha' : X \to D$ such that $\alpha'(y) = \alpha(x)$ if $y \neq x$ and $\alpha'(x) = d$.

In Definition 2.9, binding algebras are defined. Rules 1, 2 and 3 make precise what the intended interpretation of binding sorts and binding operators are. Rule 1 and 3 are familiar from ordinary many-sorted algebras. Rule 2 shows that binding sorts other than base sorts are interpreted as sets of functions. Rule 4 makes the above-mentioned condition on the interpretability of terms in binding algebras precise. In the terminology of [2], this condition is equivalent to an explicit closure condition for the interpretations of the binding sorts, and an $F$-functional condition for the interpretation of each binding operator (here $F = \langle D_s \rangle_{s \in B(S)}$).

Definition 2.9 Let $\Sigma = (S, O)$ be a binding signature. Then a binding algebra with signature $\Sigma$ is a pair $A = (\langle D_s \rangle_{s \in B(S)}, \langle T_o \rangle_{o \in O})$, where
1. For each \( s \in B(S) \), \( D_s \) is a non-empty set, called the carrier set for \( s \);
2. For each \( s \in B(S) \setminus S \), \( s = s_1, \ldots, s_n \), \( D_s \subseteq D_{s_1} \times \cdots \times D_{s_n} \rightarrow D_s \);
3. For each \( o \in O \), \( o : s_1 \times \cdots \times s_n \rightarrow s \), \( I_o \) is a function \( I_o : D_{s_1} \times \cdots \times D_{s_n} \rightarrow D_s \), called the denotation of \( o \);
4. There exists a family of functions \( \llbracket . \rrbracket : \mathcal{T}_\Sigma \rightarrow \mathcal{D} \rightarrow D_s \{ s \in B(S) \} \), where \( D = \{ D_s \}_{s \in B(S)} \), such that for all terms and assignments (writing \( [t]_\alpha \) for \( [t]_{\alpha_{s(t)}} \)):
   (a) \( [x]_\alpha = \alpha(x) \);
   (b) \( [x(t_1, \ldots, t_n)]_\alpha = \alpha(x)([t_1]_\alpha, \ldots, [t_n]_\alpha) \);
   (c) \( [x_1, \ldots, x_n \cdot t]_\alpha \) is the function \( f : D_{s(x_1)} \times \cdots \times D_{s(x_n)} \rightarrow D_s(t) \) such that,
       for all \( d_1 \in D_{s(x_1)}, \ldots, d_n \in D_{s(x_n)} \), \( f(d_1, \ldots, d_n) = [t]_{\alpha(x_1 \rightarrow d_1) \cdots (x_n \rightarrow d_n)} \);
   (d) \( [o(t_1, \ldots, t_n)]_\alpha = I_o([t_1]_\alpha, \ldots, [t_n]_\alpha) \).

For a given binding algebra \( A = (D, I) \), \( \llbracket . \rrbracket \) is uniquely determined and is called the family of evaluation functions associated with \( A \). Furthermore, an assignment \( \alpha : X \rightarrow D \), where \( X \) is a variable domain for \( \Sigma \), is called an assignment in \( A \).

Interesting among the binding algebras for a given signature are the term algebras. For their construction, we need an equivalence relation on the binding terms over the binding signature concerned. Therefore, we first define this equivalence relation. It is a version of \( \alpha \)-conversion. It identifies terms that can be obtained from each other by a change of bound variables. Notice that this equivalence relation is a congruence by construction.

**Definition 2.10** Let \( \Sigma = (S, O) \) be a binding signature and \( X \) be a variable domain for \( \Sigma \). Then \( \approx_s \) is the family of least equivalence relations \( \{ \approx_s \subseteq \mathcal{T}_\Sigma(X)_s \times \mathcal{T}_\Sigma(X)_s \}_{s \in B(S)} \) such that, writing \( t \approx t' \) for \( t \approx_{s(t)} t' \):

1. \( x_1, \ldots, x_n \cdot t \approx y_1, \ldots, y_n \cdot t[y_1, \ldots, y_n / x_1, \ldots, x_n] \)
   (for mutually distinct variables \( y_1, \ldots, y_n \) not free in \( t \));
2. \( x_1, \ldots, x_n \cdot t \approx y_1, \ldots, y_n \cdot t' \Rightarrow t[z_1, \ldots, z_n / x_1, \ldots, x_n] \approx t'[z_1, \ldots, z_n / y_1, \ldots, y_n] \)
   (for mutually distinct variables \( z_1, \ldots, z_n \) not free in \( x_1, \ldots, x_n \cdot t \) or \( y_1, \ldots, y_n \cdot t' \));
3. \( t \approx t' \) and \( s(x_1), \ldots, s(x_n) \in S \Rightarrow x_1, \ldots, x_n \cdot t \approx z_1, \ldots, z_n \cdot t' \);
4. \( t_1 \approx t'_1, \ldots, t_n \approx t'_n \), \( x : s(t_1), \ldots, s(t_n) \Rightarrow x(t_1, \ldots, t_n) \approx x(t'_1, \ldots, t'_n) \);
5. \( t_1 \approx t'_1, \ldots, t_n \approx t'_n \), \( o : s(t_1) \times \cdots \times s(t_n) \Rightarrow o(t_1, \ldots, t_n) \approx o(t'_1, \ldots, t'_n) \).

Furthermore, let \( t \in \mathcal{T}_\Sigma(X)_s \) and \( T \subseteq \mathcal{T}_\Sigma(X)_s \) for some \( s \in B(S) \). Then we write \( [t] \) for \( \{ t' \in \mathcal{T}_\Sigma(X)_s \mid t \approx t' \} \), and \( [T] \) for \( \{ [t] \mid t \in T \} \).

To refer to the interpretations of binding terms in (binding) term algebras, we also introduce the notation \( \bullet [t] \). The intended meaning of \( \bullet [t] \) is simply \( [t] \) if \( t \) is not of the form \( x_1, \ldots, x_n \cdot t' \). The intended meaning of \( \bullet [x_1, \ldots, x_n \cdot t'] \) is a function on sets of \( \approx \)-equivalence classes of terms, viz. the function of which application corresponds to taking the \( \approx \)-equivalence class of the term obtained by substitution of representatives of the arguments concerned for \( x_1, \ldots, x_n \) in \( t' \).
Definition 2.11 Let \( \Sigma = (S, O) \) be a binding signature and \( X \) be a variable domain for \( \Sigma \). Furthermore, let \( t \in T_\Sigma(X) \), for some \( s \in S \) and let \( x_1, \ldots, x_n \). \( t' \in T_\Sigma(X)_s \) for some \( s \in B(S) \setminus S \). Then we write \( \bullet_{[t]} \) for \( [t] \), and \( \bullet_{[x_1, \ldots, x_n, t']} \) for the function \( f : [T_\Sigma(X)_{s(x_1)}] \times \cdots \times [T_\Sigma(X)_{s(x_n)}] \rightarrow [T_\Sigma(X)_{s(t)}] \) such that for all \( t_1 \in T_\Sigma(X)_{s(x_1)}, \ldots, t_n \in T_\Sigma(X)_{s(x_n)} \) we have \( f([t_1], \ldots, [t_n]) = [t'[t_1, \ldots, t_n/x_1, \ldots, x_n]] \).

Notice that the function denoted by \( \bullet_{[x_1, \ldots, x_n, t']} \) is well-defined because \( \approx \) is a congruence. In Definition 2.12, term algebras are defined. Base sorts are interpreted as sets of \( \approx \)-equivalence classes of terms. Binding sorts other than base sorts are interpreted as sets of functions of the application-by-substitution kind described above on these sets of \( \approx \)-equivalence classes of terms. Binding operators are interpreted as functions on these sets of either \( \approx \)-equivalence classes or functions.

Definition 2.12 Let \( \Sigma = (S, O) \) be a binding signature and \( X \) be a variable domain for \( \Sigma \). Then the binding algebra of terms with signature \( \Sigma \) on \( X \) is the binding algebra \( A = \langle (\mathcal{D}_s)_{s \in B(S)}, (\mathcal{T}_o)_{o \in O} \rangle \), where

1. for each \( s \in B(S) \), \( \mathcal{D}_s = \{ \bullet_{[t]} \mid t \in T_\Sigma(X)_s \} \);
2. for each \( o \in O \), \( o : s_1 \times \cdots \times s_n \rightarrow s \),
   \( \mathcal{T}_o \) is the function \( F : [T_\Sigma(X)_{s_1}] \times \cdots \times [T_\Sigma(X)_{s_n}] \rightarrow [T_\Sigma(X)_s] \) such that
   \( \forall t_1 \in T_\Sigma(X)_{s_1}, \ldots, t_n \in T_\Sigma(X)_{s_n} \bullet F([t_1], \ldots, [t_n]) = [o(t_1, \ldots, t_n)] \).

The binding algebra of terms with signature \( \Sigma \) on \( X \) is the free algebra with signature \( \Sigma \) on \( X \). It is the initial algebra with signature \( \Sigma \) if \( X = \mathcal{X}_\Sigma \).

The following definition shows that substitutions of binding terms over \( \Sigma \) and \( X \) for variables in \( X \) are closely related to assignments in the binding algebra of terms with signature \( \Sigma \) on \( X \).

Definition 2.13 Let \( \Sigma = (S, O) \) be a binding signature and \( X \) be a variable domain for \( \Sigma \). Furthermore, let \( \llbracket \cdot \rrbracket \) be the family of evaluation functions associated with the binding algebra of terms with signature \( \Sigma \) on \( X \). Then a substitution \( \sigma : X \rightarrow T_\Sigma(X) \) of terms in \( T_\Sigma(X) \) for variables in \( X \) is a \( B(S) \)-indexed family of functions \( \langle \sigma_s : X_s \rightarrow T_\Sigma(X)_s \rangle_{s \in B(S)} \). The extension of \( \sigma \) from variables to terms is a \( B(S) \)-indexed family of functions \( \langle \sigma_s : T_\Sigma(X)_s \rightarrow T_\Sigma(X)_s \rangle_{s \in B(S)} \) such that for all \( s \in B(S) \) and \( t \in T_\Sigma(X)_s \) we have \( [\sigma_s(t)] = [\llbracket t \rrbracket]_{\alpha_s} \), where \( \alpha \) is the assignment \( \alpha : X \rightarrow \{ \{ \bullet_{[t]} \mid t \in T_\Sigma(X)_s \} \}_{s \in B(S)} \) such that for all \( s \in B(S) \) and \( x \in X_s \) we have \( \alpha(x) = \bullet_{[\sigma_s(x)]} \).

For every substitution \( \sigma : X \rightarrow T_\Sigma(X) \) and \( t \in T_\Sigma(X)_s \), we write \( \sigma(t) \) or \( t \sigma \) for \( \sigma_s(t) \). A substitution \( \sigma : X \rightarrow T_\Sigma(X) \) is closed if \( \sigma(x) \in C T_\Sigma(X) \) for all \( x \in X \). As in case of ordinary substitutions, we write \([t_1, \ldots, t_n/x_1, \ldots, x_n] \) for the substitution \( \sigma \) such that \( \sigma(x_1) = t_1, \ldots, \sigma(x_n) = t_n \) and \( \sigma(x) = x \) if \( x \notin \{ x_1, \ldots, x_n \} \).

Substitution extends ordinary substitution from ordinary variables to all variables. Notice that substitution is defined up to \( \approx \)-equivalence explicitly. Notice further that, if \( \sigma(x) = x_1, \ldots, x_n, t \), \( \sigma(x_{t_1}, \ldots, t_n) \) \( \approx \) \( t[\sigma(t_1), \ldots, \sigma(t_n)/x_{t_1}, \ldots, x_{t_n}] \)

Example 2.14 Other instances of binding terms involving the recursion operator of CCS are \( \mu x . z(x) \) and \( z(\mu x . z(x)) \), where \( x : P \) and \( z : P \cdot P \) (\( P \) is the sort of processes). Substitution of \( x \cdot ax \) for \( z \) in these terms yields the following results: \( \mu x . z(x)[x \cdot ax/z] = \mu x . ax \) and \( z(\mu x . z(x))[x \cdot ax/z] = \mu x . ax \).
2.3 Transition system specifications

In this subsection, we generalize the notion of TSS from conventional terms to binding terms. The meaning of TSSs is discussed in Section 3.

The TSSs of [20], which originate from [29], define binary transition relations by means of transition rules with positive premises. An extension to transition rules with positive and negative premises was presented in [18, 7] and a further extension to the specification of unary and binary relations was presented in [31]. In all three cases, variable binding operators and many-sortedness are not supported. Such an extension was first proposed in [14]. An important difference between that extension and the one presented here is that in the latter distinction between formal and actual variables, formal and actual terms, formal and actual substitutions, etc. is not made. This leads to TSSs that are more closely related to the original ones than the TSSs of [14].

We define TSSs in terms of transition rules and transition rules in terms of transition formulas. We define transition formulas over a binding signature and a domain of transition predicates. Therefore, we first define the notion of domain of transition predicates.

We assume a B(S)^*-'indexed family of mutually disjoint sets of predicates \( \mathcal{P} = \langle \mathcal{P}_r \rangle_{r \in B(S)^*} \). Furthermore, it is assumed that \( \bigcup_{r \in B(S)} V_s, \bigcup_{r \in B(S)^*} O_r \) and \( \bigcup_{r \in B(S)^*} \mathcal{P}_r \) are mutually disjoint. We write \( p : s_1 \times \ldots \times s_n \) to indicate that \( p \in \mathcal{P}_{\langle s_1, \ldots, s_n \rangle} \).

A predicate \( p : s_1 \times \ldots \times s_n \) has \( n \) arguments. If \( s_i = s_{i1}, \ldots, s_{in_i} \cdot s_i \) (\( 1 \leq i \leq n \)), then it binds \( n_i \) variables, of base sorts \( s_{i1}, \ldots, s_{in_i} \) in the \( i \)-th argument. Otherwise, i.e. if \( s_i \in S \), it does not bind any variable in the \( i \)-th argument.

**Definition 2.15** Let \( \Sigma = (S, O) \) be a binding signature. Then a domain of transition predicates on terms over \( \Sigma \) is a set \( \Pi \subseteq \mathcal{P} \) such that for all \( p \in \Pi \), if \( p : s_1 \times \ldots \times s_n \), then \( s_1, \ldots, s_n \in B(S) \), \( s_1 \in S \) and \( 1 \leq n \leq 2 \). A transition predicate \( p \in \Pi \) is called ordinary if \( p : s_1 \times \ldots \times s_n \) for some \( s_1, \ldots, s_n \in S \).

Just as in [31], we consider both unary and binary predicates as transition predicates. The restriction that a transition predicate is a unary or binary predicate is formulated here to anticipate its relaxation in Section 5. We do not consider predicates that bind variables in their first argument as transition predicates. The main reason for this exclusion is that we can not conceive of an obvious generalization of the notion of bisimulation in case variables are bound in the first argument.

Next, in Definition 2.16, we define the notions of positive and negative transition formula. We also introduce the notion of denial of a transition formula and make the notion of closed transition formula precise. Like in [18, 7] and [31], we consider both positive and negative transition formulas. The formation rule for negative formulas does not allow a negative formula of the form \( \neg p(t_1) \) for binary predicates \( p \), i.e. predicates \( p \) with \( p : s_1 \times s_2 \) for some sorts \( s_1 \) and \( s_2 \). In [18, 7] and [31], such expressions are considered to be negative formulas. We consider them to be sets of negative formulas (see also Definition 2.20).

**Definition 2.16** Let \( \Pi \) be a domain of transition predicates on terms over binding signature \( \Sigma = (S, O) \). Then \( \mathcal{F}_{\Sigma, \Pi}^+ \), the set of positive transition formulas over signature \( \Sigma \) and transition predicates \( \Pi \), is inductively defined by the following formation rule (\( 1 \leq n \leq 2 \)): 

...
and \( \mathcal{F}_{E,II}^- \), the set of negative transition formulas over signature \( \Sigma \) and transition predicates \( II \), is inductively defined by the following formation rule \((1 \leq n \leq 2)\):

\[
p \in II, p : s_1 \times \ldots \times s_n, t_1 \in \mathcal{T}_{E,s_1}, \ldots, t_n \in \mathcal{T}_{E,s_n} \Rightarrow \neg p(t_1, \ldots, t_n) \in \mathcal{F}_{E,II}^-.
\]

We use in general postfix notation for unary predicates and infix notation for binary predicates.

We write \( \mathcal{F}_{E,II} \) for \( \mathcal{F}_{E,II}^+ \cup \mathcal{F}_{E,II}^- \). For \( \phi \in \mathcal{F}_{E,II} \), \( \overline{\phi} \), the denial of \( \phi \), is defined as follows:

1. \( \overline{p(t_1, \ldots, t_m)} = \neg p(t_1, \ldots, t_m) \);
2. \( \overline{\neg p(t_1, \ldots, t_m)} = p(t_1, \ldots, t_m) \).

A positive or negative transition formula \( \phi \) is closed if all terms occurring in it are closed. We write \( \mathcal{C} \mathcal{F}_{E,II}^+ \) for \( \{ \phi \in \mathcal{F}_{E,II}^+ \mid \phi \) is closed\} and \( \mathcal{C} \mathcal{F}_{E,II}^- \) for \( \{ \phi \in \mathcal{F}_{E,II}^- \mid \phi \) is closed\}. Furthermore, we write \( \mathcal{C} \mathcal{F}_{E,II} \) for \( \mathcal{C} \mathcal{F}_{E,II}^+ \cup \mathcal{C} \mathcal{F}_{E,II}^- \).

**Example 2.17** In Example 2.5, we used \( “ x \cdot ax \) as an example of a binding term. Suppose that we have transition predicates \( \xrightarrow{a} : \mathbb{P} \times \mathbb{P} \) (one for each action \( a \)). The intended meaning of a transition formula of the form \( t \xrightarrow{a} t' \) can be explained as follows: process \( t \) is capable of first performing action \( a \) and then proceeding as process \( t' \). Hence, the transition formula \( “ x \cdot ax \xrightarrow{a} \u201c x \cdot ax \) expresses that \( “ x \cdot ax \) is capable of performing action \( a \) forever.

In the following definition, the notion of transition rule is defined. Like in [18, 7] and [31], negative formulas are not allowed as conclusions of transition rules. The notions of substitution instance and closed substitution instance of a transition rule are also introduced.

**Definition 2.18** Let \( II \) be a domain of transition predicates on terms over binding signature \( \Sigma = (S, O) \). Then \( \mathcal{R}_{E,II} \), the set of transition rules over signature \( \Sigma \) and predicates \( II \), is inductively defined by the following formation rule:

\[
\Phi \subseteq \mathcal{F}_{E,II}, \psi \in \mathcal{F}_{E,II}^+ \Rightarrow \frac{\psi}{\Phi} \in \mathcal{R}_{E,II}.
\]

Let \( r = \frac{\Phi}{\psi} \) be a transition rule. Then the transition formulas in \( \Phi \) are the premises of \( r \) and the transition formula \( \psi \) is the conclusion of \( r \). A transition rule \( r \) is closed if all formulas occurring in it are closed. Substitution extends from terms to formulas and rules as expected. For every substitution \( \sigma : \mathcal{V} \rightarrow \mathcal{T}_{\Sigma} \) and transition rule \( r \), the transition rule \( \sigma(r) \) is a substitution instance of \( r \). If \( \sigma \) is a closed substitution, the transition rule \( \sigma(r) \) is a closed substitution instance of \( r \). We write \( \text{instances}(r) \) for the set of all substitution instances of \( r \), and \( \text{cinstances}(r) \) for the set of all closed substitution instances of \( r \).

**Example 2.19** The transition rule for the recursion operator of CCS is as follows:

\[
\frac{z(\mu x \cdot z(x)) \xrightarrow{a} x'}{\mu x \cdot z(x) \xrightarrow{a} x'}
\]
Finally, the notion of TSS is defined. The main difference with the original notion of TSS is that binding terms are used instead of conventional terms. This means not only that operators may bind variables in their arguments, but also that variables may have arguments. Having variables that may have arguments obviates the need to distinguish two kinds of variables, terms, substitutions, etc. in TSSs, like in [14].

**Definition 2.20** A transition system specification (TSS) is a triple $P = (\Sigma, \Pi, R)$, where

1. $\Sigma$ is a binding signature;
2. $\Pi$ is a domain of transition predicates on terms over $\Sigma$;
3. $R \subseteq \mathcal{R}_{\Sigma, \Pi}$.

A TSS is positive if all premises of its transition rules are positive transition formulas. We write $\text{instances}(R)$ for the set $\{\text{instances}(r) \mid r \in R\}$, and $\text{cinstances}(R)$ for the set $\{\text{cinstances}(r) \mid r \in R\}$.

For each $p \in \Pi$, $p : s_1 \times s_2$, and $t_1 \in T_{s_1}$, we write $\neg p(t_1)$ for the set of formulas $\{\neg p(t_1, t_2) \mid t_2 \in CT_{s_2}\}$.

Recall that, unlike in [18, 7] and [31], an expression of the form $\neg p(t_1)$ is not considered to be a negative formula if $p$ is a binary predicate. Instead, it is considered to be an abbreviation of the set of formulas that contains all formulas $\neg p(t_1, t_2)$ where $t_2$ is a closed term of the appropriate sort. This leads to some simpler definitions in the remaining sections.

**Example 2.21** We consider a fragment of CCS without restriction and relabeling, but with recursion. CCS assumes a set $N$ of names. The set $A$ of actions is defined by $A = N \cup \bar{N} \cup \{\tau\}$, where $\bar{N} = \{\bar{a} \mid a \in N\}$. Elements $\bar{a} \in \bar{N}$ are called co-names and $\tau$ is called the silent step. The signature of the TSS for this fragment of CCS consists of a constant $0 : \rightarrow P$, a unary operator $\alpha : P \rightarrow P$ for each $\alpha \in A$, binary operators $+: P \times P \rightarrow P$ and $\mid : P \times P \rightarrow P$, and a unary variable binding operator $\mu : P \rightarrow P$. The transition predicate domain consists of a binary transition predicate $\rightarrow_{\alpha} : P \times P$ for each $\alpha \in A$. The transition rules are the transition rules given below and the transition rule given in Example 2.19 ($\alpha \in A, a \in N$).

\[
\begin{align*}
\text{Rule} & \quad & \text{Formula} \\
\text{1.} & \quad & \frac{ax}{x} \quad \frac{x \rightarrow x'}{x + y \rightarrow x'} \\
\text{2.} & \quad & \frac{x \rightarrow x'}{x | y \rightarrow x' | y} \quad \frac{y \rightarrow y'}{x | y \rightarrow x | y'} \\
\text{3.} & \quad & \frac{x \rightarrow x', y \rightarrow y'}{x \rightarrow x', y \rightarrow y'} \quad \frac{x \rightarrow x', y \rightarrow y'}{x \rightarrow x', y \rightarrow y'} \quad \frac{x \rightarrow x', y \rightarrow y'}{x \rightarrow x', y \rightarrow y'}
\end{align*}
\]

3 The meaning of TSSs

In this section, we first introduce the basic notions relevant to the issue of associating models with TSSs. These basic notions are sufficient in case of positive TSSs. Next, we discuss the principle underlying the association of a model with a positive TSS. Finally, using this principle as a guideline, we introduce the additional notions relevant to the issue of associating models with TSSs that are not positive.

The meaning of TSSs with negative premises has been extensively studied for TSSs that define transition relations on conventional terms, see e.g. [1], [18], [31], [7] and
Most definitions and results generalize naturally to the case of TSSs that define
transition relations on $\approx$-equivalence classes of binding terms, which will henceforth
often loosely be referred to as transition relations on binding terms. However, the
smooth generalization is perhaps not apparent because the presentation of the material
is new here. All work on the meaning of TSSs with negative premises uses many results
of work on logic programming with negation. An excellent survey of relevant work in
that area is [3].

3.1 Supported models

We define the notions of model of a TSS and interpretation supported by a TSS.
This requires to introduce first a kind of structures in which transition rules can be
interpreted. We introduce two equivalent kinds: transition systems and Herbrand
interpretations. However, in the sequel, we will focus on Herbrand interpretations.

The most obvious choice of structures for the interpretation of transition rules is
probably the choice of transition systems. They are defined here with respect to a
TSS, but they can alternatively be defined with respect to a binding signature and a
domain of transition predicates.

Definition 3.1 Let $P = (\Sigma, \Pi, R)$ be a TSS. A transition system for $P$ is a family
of relations $J = \langle J_p \rangle_{p \in \Pi}$, where

for each $p \in \Pi$, $p : s_1 \times \cdots \times s_n$, $J_p$ is a relation $J_p \subseteq [CT_{S_1}] \times \cdots \times [CT_{S_n}]$,
called the denotation of $p$.

So predicates are interpreted as relations on sets of $\approx$-equivalence classes of closed
binding terms. This makes precise that we identify binding terms that can be obtained
from each other by change of bound variables. This identification of binding terms
induces the following identification of transition formulas.

Definition 3.2 Let $\Pi$ be a domain of transition predicates on terms over binding
signature $\Sigma = (S, O)$. Then $\approx$ is the least equivalence relation $\approx \subseteq F_{\Sigma, \Pi} \times F_{\Sigma, \Pi}$ such that

$t_1 \approx t'_1, \ldots, t_n \approx t'_n, p : s(t_1) \times \cdots \times s(t_n) \Rightarrow
p(t_1, \ldots, t_n) \approx p(t'_1, \ldots, t'_n), \neg p(t_1, \ldots, t_n) \approx \neg p(t'_1, \ldots, t'_n)$.

Furthermore, let $\phi \in F_{\Sigma, \Pi}$ and $\Phi \subseteq F_{\Sigma, \Pi}$. Then we write $[\phi]$ for $\{\psi \in F_{\Sigma, \Pi} \mid \phi \approx \psi\}$, and $[\Phi]$ for $\{[\phi] \mid \phi \in \Phi\}$.

Another choice of structures for the interpretation of transition rules, customary in
logic programming, is the choice of Herbrand interpretations.

Definition 3.3 Let $P = (\Sigma, \Pi, R)$ be a TSS. A Herbrand interpretation for $P$ is a
set $M \subseteq C F_{\Sigma, \Pi}$ such that $[M] \subseteq M$.

An Herbrand interpretation is simply a set of transition formulas. The condition
$[M] \subseteq M$ implies that either all transition formulas from the same $\approx$-equivalence
class are in a Herbrand interpretation or none is. Thus, it is guaranteed that there
exists a bijection between the class of Herbrand interpretations for $P$ and the class of
transition systems for $P$: a Herbrand interpretation $M$ corresponds to the transition
system $J = \langle J_p \rangle_{p \in \Pi}$ such that
for each $p \in \Pi$, $\mathcal{J}_p = \{([t_1], \ldots, [t_n]) \mid p(t_1, \ldots, t_n) \in \mathcal{M}\}$.

Hence, a transition relation on binding terms over $\Sigma$ can be regarded as a set of closed positive transition formulas over $\Sigma$. Therefore, closed positive transition formulas will sometimes loosely be referred to as transitions. Likewise, closed negative transition formulas will sometimes be referred to as negative transitions. Because of the existence of a bijection between the class of Herbrand interpretations for a TSS and the class of transition systems for it, we can safely focus on Herbrand interpretations. The latter structures make it easier to explain what model is associated with a TSS that is not positive.

Before we can define what Herbrand interpretations for a TSS are models of that TSS and what Herbrand interpretations for a TSS are supported by that TSS, we have to make precise what it means for a transition formula to hold in a Herbrand interpretation. The following definition states that a positive transition formula holds in a Herbrand interpretation if it is contained in that Herbrand interpretation and a negative transition formula holds in a Herbrand interpretation if its denial is not contained in that Herbrand interpretation.

**Definition 3.4** Let $\mathcal{M}$ be a Herbrand interpretation for a TSS $P = (\Sigma, \Pi, R)$. Furthermore, let $\phi \in \mathcal{C}_E,\Pi$. Then $\phi$ holds in $\mathcal{M}$, written $\mathcal{M} \models \phi$, if

1. either $\phi \in \mathcal{C}_E,\Pi^+$ and $\phi \in \mathcal{M}$;
2. or $\phi \in \mathcal{C}_E,\Pi^-$ and $\overline{\phi} \notin \mathcal{M}$.

For $\Phi \subseteq \mathcal{C}_E,\Pi$, we write $\mathcal{M} \models \Phi$ to indicate that $\mathcal{M} \models \phi$ for all $\phi \in \Phi$.

Notice that $\mathcal{M} \models \phi$ iff $\mathcal{M} \models |\phi|$. Next, in Definitions 3.5 and 3.6, we define the notions of model of a TSS and interpretation supported by a TSS. Roughly speaking, these definitions express that a Herbrand interpretation for a TSS is a model of that TSS if it obeys the transition rules of the TSS and that a Herbrand interpretation for a TSS is supported by that TSS if all transitions contained in it are justified by the transition rules of the TSS.

**Definition 3.5** Let $\mathcal{M}$ be a Herbrand interpretation for a TSS $P = (\Sigma, \Pi, R)$. Then $\mathcal{M}$ is a Herbrand model of $P$, written $\mathcal{M} \models P$, if for all $\psi \in \mathcal{C}_E,\Pi^+$:

$$\mathcal{M} \models \psi \iff \exists \frac{\phi}{\psi} \in \text{cinstances}(R) \cdot \mathcal{M} \models \Phi.$$ 

**Definition 3.6** Let $\mathcal{M}$ be a Herbrand interpretation for a TSS $P = (\Sigma, \Pi, R)$. Then $\mathcal{M}$ is supported by $P$ if for all $\psi \in \mathcal{C}_E,\Pi^+$:

$$\mathcal{M} \models \psi \Rightarrow \exists \frac{\phi}{\psi} \in \text{cinstances}(R) \cdot \mathcal{M} \models \Phi.$$ 

Finally, we define what Herbrand interpretations for a TSS agree with that TSS. This notion of agreeing with a TSS, which is used in e.g. [18, 31, 17], is also known as "being a supported model of a TSS".

**Definition 3.7** Let $\mathcal{M}$ be a Herbrand interpretation for a TSS $P = (\Sigma, \Pi, R)$. Then $\mathcal{M}$ agrees with $P$ if $\mathcal{M}$ is a Herbrand model of $P$ and $\mathcal{M}$ is supported by $P$. 

12
Example 3.8 Consider the simple TSS with a signature consisting of a constant $c$ only, two transition predicates $\xrightarrow{a}$ and $\xrightarrow{b}$, and the following transition rules:

\[
\begin{align*}
\frac{c \xrightarrow{a} c}{c} & \quad \frac{c \xrightarrow{a} c}{c} \\
\frac{c \xrightarrow{b} c}{c} & \quad \frac{c \xrightarrow{a} c}{c}
\end{align*}
\]

As usual, we use the notation $c \xrightarrow{a}$ instead of $\neg(c \xrightarrow{a})$. Both $\{c \xrightarrow{a} c\}$ and $\{c \xrightarrow{b} c\}$ agree with this TSS.

In case of positive TSSs, the notion of agreeing with a TSS has two interesting alternative characterizations, which will be given in Section 3.2.

### 3.2 Proofs and positive TSSs

We define a general notion of proof from a TSS by allowing to prove transition rules. The proof of a transition rule $\frac{\Phi}{\psi}$ corresponds to the proof of the transition formula $\psi$ under the assumptions $\Phi$. In Section 3.3, it happens that allowing to prove transition rules is quite useful.

**Definition 3.9** Let $P = (\Sigma, \Pi, R)$ be a TSS. Then a *proof* of a transition rule $\frac{\Phi}{\psi}$ from $P$ is a well-founded, upwardly branching tree of which the nodes are labelled by formulas in $\mathcal{F}_{\Sigma, \Pi}$, such that

1. the root is labelled by $\psi$;
2. if a node is labelled by $\phi$ and $\Phi'$ is the set of labels of the nodes directly above this node, then there exists a $\psi'$ such that $\phi \approx \psi'$ and

   a) either $\psi' \in \Phi$ and $\Phi' = \emptyset$,

   b) or $\frac{\Phi'}{\psi'} \in \text{instances}(R)$.

A transition rule $r$ is *provable* from $P$, written $P \vdash r$, if there exists a proof of $r$ from $P$. A positive transition formula $\phi$ is *provable* from $P$, written $P \vdash \phi$, if there exists a proof of $\frac{\Phi}{\phi}$ from $P$.

**Example 3.10** In Example 2.17, we used $\mu x . ax \xrightarrow{a} \mu x . ax$ as an example of a transition formula. The proof of $\mu x . ax \xrightarrow{a} \mu x . ax$ is as follows:

\[
\begin{array}{c}
\mu x . ax \xrightarrow{a} \mu x . ax \\
\end{array}
\]

The non-root node is obtained from the first rule given in Example 2.21 with substitution of $\mu x . ax$ for $x$. The root node is obtained from the non-root node and the rule for the recursion operator given in Example 2.19 with substitution of $x . ax$ for $z$ and $\mu x . ax$ for $x'$ (see also Example 2.14).

It is easy to see that, if $\phi \in \Phi \iff \exists \phi' \in \Phi' \cdot \phi \approx \phi'$, $\phi' \in \Phi' \iff \exists \phi \in \Phi \cdot \phi \approx \phi$ and $\psi \approx \psi'$, then $P \vdash \frac{\Phi}{\psi} \iff P \vdash \frac{\Phi'}{\psi'}$. We have the following soundness result:
If $P \vdash \phi$, then for all Herbrand models $M$ of $P$, $M \models \phi \Rightarrow M \models \psi$.

The intended model of a positive TSS reflects the idea that the following principle implicitly applies to a TSS: “the only transition formulas that hold in the intended model are those derivable from the transition rules”.

**Definition 3.11** The intended Herbrand model of a positive TSS $P = (\Sigma, \Pi, R)$, written $M_P$, is the Herbrand model $\{\phi \in \mathcal{CF}^+_{\Sigma, \Pi} \mid P \vdash \phi\}$.

Clearly, every positive TSS has a unique intended Herbrand model. Moreover, the intended Herbrand model of a positive TSS is supported by that TSS. That is, the unique intended Herbrand model $M_P$ of a positive TSS $P$ is the unique Herbrand interpretation that agrees with $P$. The intended model of a positive TSS can also be characterized by means of the immediate consequence operator originating from [10]. The immediate consequence operator for a TSS $P$ applied to a Herbrand interpretation $M$ yields the smallest Herbrand interpretation containing all closed positive transition formulas that are immediate consequences of the closed transition formulas that hold in $M$ and the transition rules of $P$.

**Definition 3.12** Let $P = (\Sigma, \Pi, R)$ be a TSS. Then the immediate consequence operator for $P$ is the unary function $T_P$ on Herbrand interpretations for $P$ such that for all Herbrand interpretations $M$ for $P$:

$$T_P(M) = \bigcup \{[\psi] \mid \psi \in \mathcal{CF}^+_{\Sigma, \Pi} \text{ and } \exists \phi \in \text{instances}(R) \cdot M \models \phi\}.$$ 

For all Herbrand interpretations $M$ for a positive TSS $P = (\Sigma, \Pi, R)$:

1. $T_P(M) \subseteq M \Leftrightarrow M$ is a Herbrand model of $P$,
2. $T_P(M_P) = M_P$;

i.e. $M$ is a Herbrand model of $P$ iff $M$ is closed under $T_P$ and the intended Herbrand model $M_P$ is the least Herbrand interpretation closed under $T_P$ – and therefore supported by $P$. In case $P$ is not positive, the existence of a least Herbrand interpretation closed under $T_P$ – and consequently of a supported model of $P$ – is not guaranteed.

### 3.3 Stable models

In case a TSS is not positive, it is possible that proofs exist for transition rules $\frac{N}{\psi}$, where $N$ is a non-empty set of closed negative transition formulas and $\psi$ is a closed positive transition formula. Such proofs can never be extended to proofs of the conclusions concerned because no rule of a TSS has a negative transition formula as its conclusion. This means that for the intended model of a TSS that is not positive, it is reasonable to adapt the principle that implicitly applies to a TSS as follows: “the only transition formulas that hold in the intended model are those derivable from the transition rules under assumption of negative transition formulas that do not lead to inconsistencies”. In order to formalize this principle, it is useful to introduce an operator that replaces in a TSS the original transition rules by the provable closed transition rules without positive premises.
Definition 3.13 Let \( P = (\Sigma, \Pi, R) \) be a TSS. Then the TSS \( P^* \) is defined as \((\Sigma, \Pi, R^*)\), where

\[
R^* = \{ \frac{N}{\psi} \mid N \subseteq CF_{\Sigma,\Pi}, \psi \in CF_{\Sigma,\Pi}^+ \text{ and } P \vdash \psi \}.
\]

Adopting the above-mentioned principle, we conclude that the intended model of a TSS \( P = (\Sigma, \Pi, R) \) must be a model \( \mathcal{M} \) such that \( T_{P^*}(\mathcal{M}) \subseteq \mathcal{M} \). Such a model is called a stable model of \( P \). Stable models were first introduced in [16] to give a semantics of logic programming with negation.

However, even in case the adapted principle is applied, it is possible that there remain closed positive transition formulas \( \phi \) for which it is not possible to decide whether \( \phi \) holds in the intended model or not. In such cases, the TSS concerned is called incomplete. Clearly, an incomplete TSS does not have a (stable) model that can be designated as its intended model. In other words, an incomplete TSS does not have a least stable model, i.e. a model \( \mathcal{M} \) such that \( T_{P^*}(\mathcal{M}) = \mathcal{M} \). Besides, an unsound transition rule, i.e. a rule with a premise that contradicts the conclusion, is simply ignored in case the adapted principle is applied. For these reasons, we use the auxiliary notion of partial Herbrand interpretation to define the intended model of a TSS that is not positive.

Example 3.14 Consider the simple TSS with a signature consisting of a constant \( c \) only, two transition predicates \( \rightarrow \) and \( \rightarrow^* \), and the following transition rules:

\[
\begin{array}{c}
c \rightarrow^* c \\
c \rightarrow c \\
c \rightarrow^* c \\
c \rightarrow c \\
\end{array}
\]

The Herbrand interpretation \( \{ c \rightarrow c \} \) is a stable model of this TSS, although the premise and conclusion of the last transition rule contradict each other.

Definition 3.15 Let \( P = (\Sigma, \Pi, R) \) be a TSS. A partial Herbrand interpretation for \( P \) is a pair \( \mathcal{M} = (\mathcal{M}^+, \mathcal{M}^-) \), with sets \( \mathcal{M}^+, \mathcal{M}^- \subseteq CF_{\Sigma,\Pi}^+ \) such that \([\mathcal{M}^+] \subseteq \mathcal{M}^+\) and \([\mathcal{M}^-] \subseteq \mathcal{M}^-\). A partial Herbrand interpretation \( \mathcal{M} \) for \( P \) is consistent if \( \mathcal{M}^+ \cap \mathcal{M}^- = \emptyset \). A partial Herbrand interpretation \( \mathcal{M} \) for \( P \) is total if \( \mathcal{M}^+ \cup \mathcal{M}^- = CF_{\Sigma,\Pi}^+ \). Let \( \mathcal{M} = (\mathcal{M}^+, \mathcal{M}^-) \) and \( \mathcal{N} = (\mathcal{N}^+, \mathcal{N}^-) \) be partial Herbrand interpretations for \( P \). Then \( \mathcal{M} \subseteq \mathcal{N} \) if \( \mathcal{M}^+ \subseteq \mathcal{N}^+ \) and \( \mathcal{M}^- \subseteq \mathcal{N}^- \).

The intuition is that the positive component \( \mathcal{M}^+ \) contains the transitions that certainly hold and the negative component \( \mathcal{M}^- \) contains the transitions that certainly do not hold. Thus, a unique partial Herbrand model can be associated with all TSSs, even the incomplete ones. The conditions \([\mathcal{M}^+] \subseteq \mathcal{M}^+\) and \([\mathcal{M}^-] \subseteq \mathcal{M}^-\) imply that either all transition formulas from the same \( \approx \)-equivalence class are in one of the two components of a partial Herbrand interpretation or none is. Obviously, every Herbrand interpretation \( \mathcal{M} \) for a TSS \( P = (\Sigma, \Pi, R) \) can be identified with the total, consistent partial Herbrand interpretation \((\mathcal{M}, CF_{\Sigma,\Pi}^+ \setminus \mathcal{M})\). The ordering \( \subseteq \) on partial Herbrand interpretations is called the information ordering on partial Herbrand interpretations: if \( \mathcal{M} \subseteq \mathcal{N} \), then the transitions about which \( \mathcal{N} \) contains status information include the transitions about which \( \mathcal{M} \) contains status information. Partial Herbrand interpretations were first introduced in [12] in the context of logic programming. They are also known as three-valued Herbrand interpretations.
We will define an immediate consequence operator on partial Herbrand interpretations as well. Before we can do so, we have to make precise what it means for a transition formula to hold in a partial Herbrand interpretation. The following definition states that a positive transition formula holds in a partial Herbrand interpretation if it is contained in the positive component of that partial Herbrand interpretation and a negative transition formula holds in a partial Herbrand interpretation if its denial is contained in the negative component of that partial Herbrand interpretation.

**Definition 3.16** Let $M = (M^+, M^-)$ be a partial Herbrand interpretation for a TSS $P = (\Sigma, \Pi, R)$. In addition, let $\phi \in CF_{\Sigma, \Pi}$. Then $\phi$ holds in $M$, written $M \models_3 \phi$, if

1. either $\phi \in CF_{\Sigma, \Pi}^+$ and $\phi \in M^+$;
2. or $\phi \in CF_{\Sigma, \Pi}^-$ and $\overline{\phi} \in M^-$. 

For $\Phi \subseteq CF_{\Sigma, \Pi}$, we write $M \models_3 \Phi$ to indicate that $M \models_3 \phi$ for all $\phi \in \Phi$. Furthermore, we write $M \models_3^\perp \Phi$ to indicate that $M \models_3 \overline{\phi}$ for some $\phi \in \Phi$.

In Definition 3.17, we define an immediate consequence operator on partial Herbrand interpretations. This operator takes into account that a TSS may be incomplete: it yields both the closed positive transition formulas of which it can be decided that they are immediate consequences and the ones of which it can be decided that they are not immediate consequences.

**Definition 3.17** Let $P = (\Sigma, \Pi, R)$ be a TSS. Then the immediate consequence operator for $P$ for partial Herbrand interpretations is the unary function $T^3_P$ on partial Herbrand interpretations for $P$ such that for all partial Herbrand interpretations $M = (M^+, M^-)$ for $P$:

$$T^3_P(M) = (N^+, N^-)$$

where

$$N^+ = \bigcup \{[\psi] | \psi \in CF_{\Sigma, \Pi}^+ \text{ and } \exists \overline{\phi} \in \text{cinstances}(R) \cdot M \models_3 \overline{\phi} \};$$

$$N^- = \bigcup \{[\psi] | \psi \in CF_{\Sigma, \Pi}^+ \text{ and } \forall \overline{\phi} \in \text{cinstances}(R) \cdot M \models_3 \overline{\phi} \}.$$ 

Using Definitions 3.13 and 3.17, we now define the counterpart of stable models for partial Herbrand interpretations.

**Definition 3.18** A partial stable model of a TSS $P = (\Sigma, \Pi, R)$, is a partial Herbrand interpretation $M$ for $P$ such that $T^3_P(M) \subseteq M$, where $\subseteq$ is the information ordering; i.e. a partial stable model is a partial Herbrand interpretation closed under $T^3_P$. There exists a unique least partial stable model for any TSS. The least partial stable model is consistent for any TSS. We write $M^3_p$ for the least partial stable model of $P$.

Because the immediate consequence operator for partial Herbrand interpretations yields consequences on the basis of what certainly holds and what certainly does not hold, this operator also prevents unsound transition rules from being unnoticed. Unsound transition rules lead to the partial stable model $(0, 0)$. Least partial stable models were shown in [27] to coincide with the well-founded models introduced earlier in [15] in the context of logic programming.

There exists a least partial stable model for every TSS. However, we are only interested in those TSSs of which the least partial stable model can be identified with a stable model.
Definition 3.19 A TSS \( P \) is meaningful if its least partial stable model \( M_P^P \) is total.

In [17], a lot of evidence is given for the claim that the definition of meaningful TSS given above is the most general one without undesirable properties. Occasionally, we may also be interested in TSSs of which the least partial stable model is not total. However, excluded are occasions on which it is essential that it can be decided for every transition whether it holds in the intended model or not. Hence, excluded are occasions on which it must be derivable from the TSS concerned whether two terms are bisimilar, such as in case of a transition rule format guaranteeing that bisimulation is a congruence. Devising a stratification [31] for a TSS is a proven technique to check whether it is meaningful.

Definition 3.20 The intended Herbrand model of a meaningful TSS \( P = (\Sigma, \Pi, R) \), written \( M_P \), is the Herbrand model of \( P \) such that \( M_P^P = (M_P, CF_{\Sigma, \Pi}^+ \setminus M_P) \). \( M_P \) is also called the Herbrand model associated with \( P \).

Clearly for positive TSSs, Definition 3.11 coincides with Definition 3.20. In other words, for positive TSSs, supported models and least stable models coincide.

Example 3.21 In Example 3.8, a TSS was given which has two minimal supported models. The intended model of that TSS is \( \{c \rightarrow^b c\} \). In Example 3.14, a TSS was given which has a stable model, although it has a transition rule with a premise contradicting the conclusion. That TSS does not have an intended model.

4 Bisimulation as a congruence

In this section, we first generalize the notion of bisimulation to TSSs that define transition relations on binding terms. Next, we generalize the transition rule format known as the panth format [31] accordingly. The main result of this paper is that the generalized panth format guarantees that generalized bisimulation is a congruence. The proof of the congruence theorem is outlined in Appendix B.

Bisimulation is a frequently used equivalence to abstract from irrelevant details of operational semantics. Originally introduced in modal logic, it was introduced in process theory in [26]. The first format guaranteeing that bisimulation is a congruence appears to be the de Simone format [29]. The original panth format generalizes the ntyft/ntyxt format of [18] for unary predicates, which in turn extends the tyft/tyxt format of [20] with negative premises. The original panth format also generalizes the path format of [6], which extends tyft/tyxt format as well.

4.1 Bisimulation

In Definition 4.1, we define the notion of bisimulation based on a TSS for TSSs that define transition relations on binding terms. Rule 1 is needed because we identify binding terms that can be obtained from each other by change of bound variables. Rule 2 and 3 are familiar from ordinary (strong) bisimulation. Rule 4 is reminiscent of the closure-under-substitutions property of open bisimulation equivalence, an equivalence proposed for the \( \pi \)-calculus in [28].
Definition 4.1 Let $P = (<\Sigma, \Pi, R>)$, where $\Sigma = (S, O)$, be a TSS. Then a bisimulation $B$ based on $P$ is a family of symmetric binary relations $\langle B_s \subseteq CT_{E_S} \times CT_{E_S} \rangle_{s \in B(S)}$ such that, writing $B(t, t')$ for $B_s(t, t')$:

1. $t \approx t' \Rightarrow B(t, t')$
2. $B(t, t')$ and $p(t_1, t_2) \in M_P \Rightarrow \exists t'_2 \cdot p(t'_1, t'_2) \in M_P$ and $B(t_2, t'_2)$
3. $B(t, t'_1)$ and $p(t_1) \in M_P \Rightarrow p(t'_1) \in M_P$
4. $B(t_1, \ldots, t_n, t_1, \ldots, y_n \cdot t') \Rightarrow \forall t_1 \in CT_{E_S(x_1)}, \ldots, t_n \in CT_{E_S(x_n)} \cdot B(t_1, \ldots, t_n/x_1, \ldots, x_n, t'[t_1, \ldots, t_n/y_1, \ldots, y_n])$.

Two closed terms $t, t' \in CT_{E_S}$ ($s \in B(S)$) are bisimilar in $P$, written $t \equiv_P t'$, if there exists a bisimulation $B$ such that $B(t, t')$.

4.2 Panth format

In Definition 4.2, we generalize the panth format of [31]. There is no essential difference between the definition of the panth format in [31] and the one given below. Rules 1, 3 and 4 are the rules as formulated in [13] for TSSs that do not support variable binding operators. However, rule 3 does not exclude terms in which variables are bound from occurring in the first argument of conclusions. Rule 2 is only needed because we chose to treat expressions of the form $\neg p(t_1)$ as abbreviations if $p$ is a binary predicate.

Definition 4.2 Let $P = (<\Sigma, \Pi, R>)$ be a TSS. Then a transition rule $r = \frac{p}{\psi} \in R$ is in panth format if it satisfies the following restrictions:

1. for each positive premise of the form $p(t_1, t_2) \in \Phi$, the second argument $t_2$ is a variable;
2. for each negative premise of the form $\neg p(t_1, t_2) \in \Phi$, the second argument $t_2$ is a closed term;
3. the conclusion $\psi$ has the form $p(t_1)$ or $p(t_1, t_2)$, where in either case the first argument $t_1$ contains at most one operator;
4. the variables that occur as second argument of a positive premise of the form $p(t_1, t_2)$ or in the first argument of the conclusion are mutually distinct.

The TSS $P$ is in panth format if each transition rule $r \in R$ is in panth format.

It follows from rule 3 of the panth format that a conclusion $\psi$ must have one of the following forms:

1. $p(o(u_1, \ldots, u_n))$ or $p(o(u_1, \ldots, u_n), t)$, where $u_i$ ($1 \leq i \leq n$) is a variable or a term of the form $x_1, \ldots, x_m \cdot x(x_1, \ldots, x_m)$;
2. $p(x(x_1, \ldots, x_n))$ or $p(x(x_1, \ldots, x_n), t)$;
3. $p(x)$ or $p(x, t)$.

In Definitions 4.3 and 4.4, we define the notion of well-founded transition rule and the related notion of pure transition rule. These notions are used in the proofs of the congruence theorems from [7] and [31]. The proofs concerned make use of the result from [13] that a TSS in ntyft/ntyxt format or panth format can be transformed into
an equivalent TSS in the format concerned with transition rules that are well-founded. This result extends to the generalized panth format presented above if only ordinary transition predicates are used, i.e. transition predicates that do not bind any variable in their arguments.

**Definition 4.3** Let $P = (\Sigma, \Pi, R)$ be a TSS. In addition let $\Phi \subseteq \mathcal{F}_{E, \Pi}^\downarrow$. Then the *variable dependency graph* of $\Phi$ is a directed unlabeled graph with the variables occurring in $\Phi$ as nodes and as edges:

$$\{(x, x') \mid \exists p(t, t') \in \Phi : x \text{ occurs in } t \text{ and } x' \text{ occurs in } t'\}.$$ 

$\Phi$ is *well-founded* if every backward chain of edges in its variable dependency graph is finite. A transition rule $r = \Phi \not\vdash \psi \in R$ is *well-founded* if the set $\{\phi \in \Phi \mid \phi \in \mathcal{F}_{E, \Pi}^\downarrow\}$ is well-founded. The TSS $P$ is *well-founded* if each transition rule $r \in R$ is well-founded.

**Definition 4.4** Let $P = (\Sigma, \Pi, R)$ be a TSS and $r \in R$ be a transition rule. A variable $x$ is *free* in $r$ if it is a free variable of some term occurring in $r$, but does not occur as second argument of a positive premise of the form $p(t_1, t_2)$ or in the first argument of the conclusion. The transition rule $r$ is *pure* if $r$ is well-founded and there are no variables free in $r$.

The following result shows that a meaningful TSS in panth format is equivalent to one that is well-founded if only ordinary transition predicates are used.

**Theorem 4.5** Let $P = (\Sigma, \Pi, R)$ be a meaningful TSS in panth format. If all transition predicates in $\Pi$ are ordinary, then there exists a well-founded TSS $P' = (\Sigma, \Pi, R')$ in panth format with $M_{P'} = M_P$.

**Proof.** The proof is outlined in Appendix B. First, we prove the case of TSSs that only define binary relations. The proof amounts to careful checking of the proof of Theorem 5.4 from [13] – a well-foundedness theorem for the ntyft/ntyxt format – and adapting it to the case with variable binding operators where needed. Next, we make use of an immediate corollary of the proof of Theorem 4.6: a TSS that defines unary relations as well can be transformed into a TSS that only defines binary relations such that the models associated with the TSSs are isomorphic.

### 4.3 Congruence theorem

The following result shows that bisimulation is a congruence if well-founded transition rules in panth format are used.

**Theorem 4.6** Let $P = (\Sigma, \Pi, R)$ be a meaningful TSS in panth format. If $P$ is well-founded, then $\equiv_P$ is a congruence.

**Proof.** The proof is outlined in Appendix B. First, we prove the case of TSSs that only define binary relations. The proof amounts to careful checking of the proof of Theorem 8.13 from [7] – the congruence theorem for the ntyft/ntyxt format – and adapting it to the case with variable binding operators where needed. Next, we show that each TSS that defines unary relations as well can be reduced to a TSS that only defines binary relations, while preserving bisimilarity. This is straightforward in the many-sorted case. □
Usually, proofs of congruence theorems are intricate. This includes the proof of the congruence theorem from [7]. Fortunately, the proof of Theorem 4.6 goes for the greater part exactly like the proof of that congruence theorem. The following is a corollary of Theorems 4.5 and 4.6.

**Corollary 4.7** Let $P = (\Sigma, \Pi, R)$ be a meaningful TSS in panth format. If all transition predicates in $\Pi$ are ordinary, then $\equiv_P$ is a congruence.

In case not all transition predicates are ordinary, well-foundedness has to be checked.

## 5 Given sorts and parametrization

In various applications of TSSs, it is impractical and unnecessary to provide the terms of certain sorts with an operational semantics because there exists a fully established semantics for them. We will call such sorts *given sorts*. It is common to identify terms of given sorts if they are semantically equivalent. This can be formalized as follows. First of all, we introduce $\bar{\approx}$, the (sort-indexed) family of least congruence relations on binding terms that includes both $\approx$ and the equivalence induced by the semantics for the terms of given sorts. Next, we replace in the definitions of Sections 3 and 4 all occurrences of $\approx$ by $\bar{\approx}$. These replacements slightly alter the notions of transition system, Herbrand interpretation, proof, partial Herbrand interpretation, and bisimulation. Indirectly, they also alter the immediate consequence operators ($T_P$ and $T_P^3$), the partial stable models for a TSS, and the intended Herbrand model of a TSS ($M_P$). In all cases, the alteration is simply that more terms are identified. Finally, we replace rule 3 in the definition of the panth format (Definition 4.2) by the following rule:

3'. the conclusion $c$ has the form $p(t_1)$ or $p(t_1, t_2)$, where in either case the first argument $t_1$ contains at most one operator $\sigma : s_1 \times \ldots \times s_n \to s$ where $s$ is not a given sort;

This modified rule permits, for given sorts $s$, that a term of sort $s$ is used where the original rule only permits that a variable of sort $s$ is used. The congruence theorem goes through in the case of TSSs with given sorts (obviously with the altered definitions of bisimulation and panth format). The proof goes like the proof of Theorem 4.6.

Distinguishing given sorts does not only make it possible to relax the panth format. It also allows for TSSs that define parametrized transition relations, where the parameters are closed terms of given sorts. Put differently, we can relax the restriction that a transition predicate $p$ is a predicate $p : s_1 \times \ldots \times s_n$ with $1 \leq n \leq 2$ to the restriction that a transition predicate $p$ is a predicate $p : s_1 \times \ldots \times s_n$ with at most two sorts among $s_1, \ldots, s_n$ that are not given sorts. Suppose that $p$ is a parametrized transition predicate $p : s_1 \times \ldots \times s_n$ and $i_1, \ldots, i_k$ ($n - 2 \leq k \leq n - 1$) are the indices of the given sorts in increasing order. Then we can take a fresh predicate $p_{t_1, \ldots, t_k}$ for each closed term $t_i$ of sort $s_{i_1}, \ldots, $ closed term $t_k$ of sort $s_{i_k}$. It is easy to see that carrying on in this way, we can reduce any TSS that defines parametrized transition relations to a TSS that only defines unparametrized — unary and binary — transition relations, while preserving bisimilarity.
Example 5.1 In Appendix A, a TSS of BPA\textsuperscript{sat} with integration is given. It is a TSS that defines parametrized transition relations. The sort \( \mathbb{R}_{\geq 0} \) of non-negative real numbers and the sort \( \mathcal{P}(\mathbb{R}_{\geq 0}) \) of sets of non-negative real numbers are considered given sorts. There is one transition rule with a negative premise. Moreover, a variable binding operator, viz. the integration operator \( \int \), is involved. There exists a stratification for this TSS and consequently it is meaningful. It is easy to see that the TSS is in the relaxed panth format. In addition, all transition predicates concerned are ordinary. Hence, \( \equiv_{\mathcal{P}} \) is a congruence.

BPA\textsuperscript{sat} was introduced in [5] as a fragment of ACP\textsuperscript{sat}, standard real time process algebra with absolute timing. In that paper, integration was added to it as well.

6 Concluding remarks

The notion of TSS was first introduced in [20] and then generalized in [18], [7], [6] and [31]. We generalized it further to cover variable binding operators and many-sortedness. We found that the notions of bisimulation and panth format generalize naturally to the generalized TSSs, and moreover that in the generalized setting bisimulation is still a congruence for meaningful TSSs in panth format. Therefore, we expect that the applicability of TSSs to provide process calculi, specification languages and programming languages with an operational semantics has been improved. The generalized TSSs can amongst other things deal with: the integration operator \( \int \) of real time ACP [4], the sum operator \( \sum \) of \( \mu \text{CRL} \) [19], and the recursion operator \( \mu \) of CSP [21] and CCS [23]. If the \( \pi \)-calculus [24] would separate names and variables, guaranteeing that distinct constants remain distinct, \( \pi \)-calculus features such as input action prefixes \( \pi (y) \) and the restriction operator \( \nu \) could be dealt with as well.

Our main motivation to take up the work presented in this paper stems directly from work on an integrated treatment of all versions of ACP with timing [5]. That work created the need of a generalization of the framework from [31] that takes variable binding operators and many-sortedness into account. The notion of TSS was already generalized to deal with variable binding operators and many-sortedness in [14]. That paper gives syntactic criteria for the (operational) conservativity property of extensions, but not for the congruence property of bisimulation equivalence. Initially, we tried to generalize the congruence result of [31] to the generalized TSSs of [14]. This turned out to be far from straightforward because of the distinction made between formal and actual variables, formal and actual terms, formal and actual substitutions, etc. In addition, it was an obstacle that there is little semantic clarification in [14] of the notations introduced. Therefore, we chose to introduce an alternative generalization. We did not give syntactic criteria for conservativity of extensions, which is important in cases where an existing calculus or language is extended with new features. We keep this topic for future research, but we expect that it is easy to generalize the relevant results of [9] or to adapt the results of [14].

References


A TSS of BPA\textsuperscript{sat} with integration

In this appendix, we give the signature, the domain of transition predicates and transition rules of BPA\textsuperscript{sat} with integration. In BPA\textsuperscript{sat}, basic standard real time process algebra with absolute timing, parallelism and communication are not considered. The integration operator \( \int \) provides for alternative composition over a continuum of alternatives. In BPA\textsuperscript{sat}, a set \( A \) of actions is assumed. \( P \) is the sort of processes.

The signature of BPA\textsuperscript{sat} consists of an urgent action constant \( \tilde{a}: \rightarrow P \) for each \( a \in A \), the urgent deadlock constant \( \tilde{\delta}: \rightarrow P \), the immediate deadlock constant \( \delta: \rightarrow P \), the alternative composition operator \( +: P \times P \rightarrow P \), the sequential composition operator \( \cdot: P \times P \rightarrow P \), the absolute delay operator \( \sigma_{\text{abs}}: \mathbb{R}_{\geq 0} \times P \rightarrow P \), and the integration (variable binding) operator \( \int: P(\mathbb{R}_{\geq 0}) \times \mathbb{R}_{\geq 0} . P \rightarrow P \).

The process \( \tilde{a} \) is only capable of performing action \( a \), immediately followed by successful termination, at time \( 0 \). The process \( \tilde{\delta} \), although existing at time \( 0 \), is incapable of doing anything. The process \( \delta \) is the process that exhibits inconsistent timing at time \( 0 \). This means that \( \delta \), different from \( \tilde{\delta} \), does not exist at time \( 0 \) and hence causes a time stop at time \( 0 \). The process \( \sigma_{\text{abs}}^p(x) \) is the process \( x \) shifted in time by \( p \). Thus, the process \( \sigma_{\text{abs}}^p(\tilde{\delta}) \) is capable of idling from time \( 0 \) upto and including time \( p \) and at time \( p \) it gets incapable of doing anything — whereas the process \( \sigma_{\text{abs}}^p(\delta) \) is only capable of idling from time \( 0 \) upto, but not including, time \( p \). So \( \sigma_{\text{abs}}^p(\delta) \) can not reach time \( p \).

We need four kinds of transition predicates:

- a predicate \( (\cdot,\cdot) \rightarrow (\cdot,\cdot) : P \times \mathbb{R}_{\geq 0} \times P \times \mathbb{R}_{\geq 0} \) for each \( a \in A \),
- a predicate \( (\cdot,\cdot) \rightarrow (\cdot,\cdot) : P \times \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \) for each \( a \in A \),
- a predicate \( (\cdot,\cdot) \rightarrow (\cdot,\cdot) : P \times P \),
- a predicate \( \text{ID}(\cdot) : P \).

The four kinds of transition predicates are called the action step, action termination, time step and immediate deadlock predicates. They can be explained as follows:

\[
\begin{align*}
(t,p) & \rightarrow (t',p): \text{process } t \text{ is capable of first performing action } a \text{ at time } p \\
& \text{and then proceeding as process } t'; \\
(t,p) & \rightarrow (\cdot, \cdot): \text{process } t \text{ is capable of first performing action } a \text{ at time } p \\
& \text{and then terminating successfully}; \\
(t,p) & \rightarrow (t',q): \text{process } t \text{ is capable of first idling from time } p \text{ to time } q \\
& \text{and then proceeding as process } t'; \\
\text{ID}(t): & \text{process } t \text{ is not capable of reaching time } 0.
\end{align*}
\]

The transition rules for BPA\textsuperscript{sat} with integration are given in Tables 1 and 2. These rules are easy to understand. We will only explain the rules for the absolute delay operator \( (\sigma_{\text{abs}}) \). The rules in the second row express that the action related capabilities of a process \( \sigma_{\text{abs}}^0(x) \) at time \( p \) include those of process \( x \) at time \( p \) and that the action related capabilities of a process \( \sigma_{\text{abs}}^p(x) \) at time \( p + r \) include those of process \( x \) at time \( p \) shifted in time by \( r \). The first and second rule in the fifth row express that a process \( \sigma_{\text{abs}}^r(x) \) \((r > 0)\) can idle from any time \( p \geq 0 \) to any time \( p' < r \) and that it can also idle to time \( r \) provided that process \( x \) can reach time \( 0 \). The third rule in the fifth row expresses that the time related capabilities of a process \( \sigma_{\text{abs}}^q(x) \) \((q \geq 0)\) at time \( p + q \) include those of process \( x \) at time \( p \) shifted in time by \( q \).
B Outline of proofs

Theorem 4.5 (Well-foundedness) Let $P = (\Sigma, \Pi, R)$ be a meaningful TSS in panth format. If all transition predicates in $\Pi$ are ordinary, then there exists a well-founded TSS $P' = (\Sigma, \Pi', R')$ in panth format with $M_{P'} = M_P$.

Proof. If Theorem 5.4 from [13] goes through for many-sorted terms with variable binding in case all transition predicates are ordinary, the theorem follows immediately. It is straightforward to check that Theorem 5.4 from [13] goes through. The proof presented in [13] makes, directly or indirectly, use of Lemma 3.2 and adaptations of Lemmas and Theorems 4.1, 4.5 and 4.9 from that paper. It is immediately clear that Lemma 3.2 goes through for many-sorted terms with variable binding seeing that its proofs do not depend on the term structure. The proof of the adaptation of Lemma 4.1 needs a slight adaptation of the substitutions because the operators may now be variable binding operators. In the proofs of the adaptations of Theorems 4.5 and 4.9, it is now necessary to make case distinctions wherever the form of the first argument of premises or conclusions matters because it may have the form $x(x_1, \ldots, x_n)$ as well. However, the additional cases are similar to the the original ones. □
Theorem 4.6 (Congruence) Let $P = (\Sigma, \Pi, R)$, where $\Sigma = (S, O)$, be a meaningful TSS in path format. If $P$ is well-founded, then $\sqsubseteq_P$ is a congruence.

Proof. First, we prove the case that all predicates in $\Pi$ are binary predicates. Next, we show that, if there are also unary predicates in $\Pi$, we can construct from the TSS $P$ a new TSS $P'$ without unary predicates such that two closed terms over $\Sigma$ are bisimilar in $P$ if and only if they are bisimilar in $P'$.

Theorem 8.13 from [7] covers the case that all predicates in $\Pi$ are binary predicates for single-sorted terms without variable binding. In that theorem the requirement of well-foundedness is omitted because it follows from Theorem 5.4 from [13] that it can be omitted. Theorem 8.13 from [7] goes through for many-sorted terms with variable binding if we add the requirement of well-foundedness. The proof presented in [7] makes, directly or indirectly, use of the Lemmas, Theorems and Corollaries 5.5–5.8, 8.6, 8.8, 8.9 and 8.12 from that paper as well as Theorem 5.4 from [13]. We do not have to check that Theorem 5.4 from [13] goes through because we do not omit the requirement of well-foundedness in our congruence theorem. It is immediately clear that Lemmas, Theorems and Corollaries 5.5–5.8 go through for many-sorted terms with variable binding seeing that their proofs do not depend on the term structure. It requires little effort to check that Lemmas 8.6, 8.8 and 8.12 go through as well. The proof of Lemma 8.6 needs a slight adaptation of the substitutions $\sigma_f$ because $f$ may now be a variable binding operator. The proof of Lemma 8.8 remains straightforward after the adaptation of $R_P$, meant to be the minimal congruence that includes $\sqsubseteq_P$, to terms in which variables are bound. The proof of Lemma 8.12 simply combines Corollary 5.8 and Lemma 8.9.

Surprisingly, it is also straightforward to check that Lemma 8.9, which is the main lemma, goes through. That lemma consist of two parts. In the proof of both parts, it is now necessary to show as well that

$$x_1, \ldots, x_n \cdot u \cdot R_P \cdot y_1, \ldots, y_n \cdot v \Rightarrow \forall t_1 \in CT_{\Sigma_s(x_1)}, \ldots, t_n \in CT_{\Sigma_s(x_n) \cdot u[t_1, \ldots, t_n/x_1, \ldots, x_n]} \cdot R_P \cdot v[t_1, \ldots, t_n/y_1, \ldots, y_n].$$

This follows immediately from the definitions of $\sqsubseteq_P$ and $R_P$. The proof of the first part makes use of Claim 8.10 and the proof of the second part makes use of Claim 8.11. Both claims concern the existence of a substitution with certain properties. In the proof of property (b) of both claims, now two cases have to be distinguished: (i) $x_i : s$ for some $s \in S$ and (ii) $x_i : s$ for some $s = s_1 \times \ldots \times s_n \cdot s \in B(S)$. The proof for case (ii) is easy using the following fact:

$$\sigma(x_i) \not\in V_s \Rightarrow \sigma(x_i) \approx \sigma(y_1, \ldots, y_n \cdot x_i(y_1, \ldots, y_n)).$$

This fact follows immediately from the definition of substitution. In the proof of both parts, it is also necessary to take into account that the last transition rule applied in a proof may have a conclusion of the form $p(x'(x_1, \ldots, x_n), t)$. This means that variants of Claims 8.10 and 8.11 have to be proved that include the additional property

$$(b') \text{ if } x = x' \text{ then } \sigma'(x) = x_1, \ldots, x_n \cdot f(x_1, \ldots, x_n),$$

which is easy to prove.

Constructing from the TSS $P$ a new TSS $P'$ without unary predicates that preserves bisimilarity is for many-sorted terms much easier than for single-sorted terms.
(see Theorem 4.5 in [31] for an appropriate construction in the single-sorted case). For many-sorted terms with variable binding, the construction of $P' = (\Sigma', \Pi', R')$, where $\Sigma' = (\mathcal{S}', \mathcal{O}')$, goes as follows:

1. $\mathcal{S}' = \mathcal{S} \cup \{s_\xi\}$, where $s_\xi \in \mathcal{S} \setminus \mathcal{S}$;
2. $\mathcal{O}' = \mathcal{O} \cup \{\xi\}$, where $\xi: \to s_\xi$;
3. $\Pi' = \{p \in \Pi \mid \exists s_1 \in \mathcal{S}, s_2 \in \mathcal{B}(\mathcal{S}) \cdot p : s_1 \times s_2\} \cup \{p_\xi \in \mathcal{P} \mid \exists p \in \Pi, s \in \mathcal{S} \cdot p : s \text{ and } p_\xi : s \times s_\xi\}$;
4. $R' = \{\{\Xi(\phi) \mid \phi \in \Phi\} \mid \phi \in R\}$,
where $\Xi(p(t_1, t_2)) = p(t_1, t_2)$, $\Xi(-p(t_1, t_2)) = -p(t_1, t_2)$, $\Xi(p(t)) = p_\xi(t, z)$ ($z$ fresh and $z \in \mathcal{V}_\xi$), $\Xi(-p(t)) = -p_\xi(t, \xi)$.

That is, $s_\xi$ is a fresh sort, $\xi$ is the only term of sort $s_\xi$, binary predicates $p_\xi$ take the place of unary predicates $p$, and the new transition rules are obtained from the old ones by replacing each positive transition formula $p(t)$ by $p_\xi(t, z)$ and each negative transition formula $-p(t)$ by $-p_\xi(t, \xi)$. The proviso that the variables $z$ used in these replacements must be fresh means that they are mutually distinct. It follows immediately that $\sigma(p_\xi(t, x)) = p_\xi(\sigma(t), \xi)$ for all closed substitutions $\sigma$. Next it is easy to see that $p(t) \in \mathcal{M}_P \iff p_\xi(t, \xi) \in \mathcal{M}_P$, $p(t_1, t_2) \in \mathcal{M}_P \iff p(t_1, t_2) \in \mathcal{M}_P'$, and $\xi \equiv_P \xi$. Consequently, for all $s \in \mathcal{B}(\mathcal{S})$, for all $t, t' \in \mathcal{C}_{\Pi, \mathcal{S}}$, $t \equiv_P t' \iff t \equiv_{P'} t'$. \qed
Computing Science Reports

Department of Mathematics and Computing Science
Eindhoven University of Technology

If you want to receive reports, send an email to: m.m.j.philips@tue.nl (we cannot guarantee the availability of the requested reports)

In this series appeared:

<table>
<thead>
<tr>
<th>Volume</th>
<th>Authors</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>96/01</td>
<td>M. Voorhoeve and T. Basten</td>
<td>Process Algebra with Autonomous Actions, p. 12.</td>
</tr>
<tr>
<td>96/02</td>
<td>P. de Bra and A. Aerts</td>
<td>Multi-User Publishing in the Web: DreSS, A Document Repository Service Station, p. 12</td>
</tr>
<tr>
<td>96/06</td>
<td>W.M.P. van der Aalst and T. Basten</td>
<td>Life-Cycle Inheritance A Petri-Net-Based Approach, p. 18.</td>
</tr>
<tr>
<td>96/07</td>
<td>M. Voorhoeve</td>
<td>Structural Petri Net Equivalence, p. 16.</td>
</tr>
<tr>
<td>96/11</td>
<td>T. Laan</td>
<td>AUTOMATH and Pure Type Systems, p. 30.</td>
</tr>
<tr>
<td>96/12</td>
<td>F. Kamareddine and T. Laan</td>
<td>A Correspondence between Nuprl and the Ramified Theory of Types, p. 12.</td>
</tr>
<tr>
<td>96/13</td>
<td>T. Borghuis</td>
<td>Priorean Tense Logics in Modal Pure Type Systems, p. 61</td>
</tr>
<tr>
<td>96/14</td>
<td>S.H.J. Bos and M.A. Reniers</td>
<td>The $I^2$ C-bus in Discrete-Time Process Algebra, p. 25.</td>
</tr>
<tr>
<td>96/15</td>
<td>M.A. Reniers and J.J. Vereijken</td>
<td>Completeness in Discrete-Time Process Algebra, p. 139.</td>
</tr>
<tr>
<td>96/17</td>
<td>E. Brönn and P. Hoogendijk</td>
<td>Nested collections and polytypism, p. 11.</td>
</tr>
<tr>
<td>96/18</td>
<td>P.D.V. van der Stok</td>
<td>Real-Time Distributed Concurrency Control Algorithms with mixed time constraints, p. 71.</td>
</tr>
<tr>
<td>96/19</td>
<td>M.A. Reniers</td>
<td>Static Semantics of Message Sequence Charts, p. 71</td>
</tr>
<tr>
<td>96/20</td>
<td>L. Feijs</td>
<td>Algebraic Specification and Simulation of Lazy Functional Programs in a concurrent Environment, p. 27.</td>
</tr>
<tr>
<td>96/23</td>
<td>W.M.P. van der Aalst</td>
<td>Structural Characterizations of sound workflow nets, p. 22.</td>
</tr>
<tr>
<td>96/24</td>
<td>M. Voorhoeve and W. van der Aalst</td>
<td>Conservative Adaption of Workflow, p.22</td>
</tr>
<tr>
<td>96/25</td>
<td>M. Vaccari and R.C. Backhouse</td>
<td>Deriving a systolic regular language recognizer, p. 28</td>
</tr>
<tr>
<td>97/06</td>
<td>M. Franssen</td>
<td>Tools for the Construction of Correct Programs: an Overview, p. 33.</td>
</tr>
<tr>
<td>97/07</td>
<td>J.C.M. Baeten and J.A. Bergstra</td>
<td>Bounded Stacks, Bags and Queues, p. 15.</td>
</tr>
</tbody>
</table>

Communication Modeling: The Language/Action Perspective, p. 147.

Distributed real-time systems: a survey of applications and a general design model, p. 31.


Exploring the Process Dimension of Workflow Management, p. 56.

A computer checked algebraic verification of a distributed summation algorithm, p. 28

XP*: A Pure Type System for First Order Logic with Automated Theorem Proving, p. 35.

On the verification of Inter-organizational workflows, p. 23

Calculating a Round-Robin Scheduler, p. 23

Informawetenschap 1997 Wetenschappelijke bijdragen aan de Vijfde Interdisciplinaire Conferentie Informatiewetenschap, p. 60


State I Event Net Equivalence, p. 25

Deadlock Behaviour in Split and ST Bisimulation Semantics, p. 15.

Pair Algebras and Galois Connections, p. 14

Flat Fragments of CTL and CTL*: Separating the Expressive and Distinguishing Powers. P. 22.

Maintenance of the Union of Intervals on a Line Revisited, p. 10.

Checking verifications of protocols and distributed systems by computer. Extended version of a tutorial at CONCUR'98, p. 27.

Structured Operational Semantics of z, p. 27

Diagnosing Workflow Processes using WoFLan, p. 44

Final Dialgebras: From Categories to Allegories, p. 26

Process Algebra with Interleaving Probabilistic Parallel Composition, p. 81
99/05 M. Franssen, R.C. Velkamp and W. Wesselink
Efficient Evaluation of Triangular B-splines, p. 13

99/06 T. Basten and W. v.d. Aalst
Inheritance of Workflows: An Approach to tackling problems related to change, p. 66

99/07 P. Brasilovsky and P. De Bra

99/08 D. Bosmacki, S. Mauw, and T. Wijlemse

99/09 J. v.d. Pol, J. Hoorn and E. de Jong
Requirements Specification and Analysis of Command and Control Systems

99/10 T.A.C. Wijlemse
The Analysis of a Conveyor Belt System, a case study in Hybrid Systems and timed \( \mu \) CRL, p. 44.

99/11 J.C.M. Baeten and C.A. Middelburg
Process Algebra with Timing: Real Time and Discrete Time, p. 50.

99/12 S. Andova

99/13 K.M. van Hee, R.A. van der Toorn, J. van der Woude and P.A.C. Verkoulen
A Framework for Component Based Software Architectures, p. 19

99/14 A. Engels and S. Mauw
Why men (and octopuses) cannot juggle a four ball cascade, p. 10

An algorithm for the asynchronous Write-All problem based on process collision*, p. 11.

99/16 G.J. Houben, P. Lemmens

99/17 T. Basten, W.M.P. v.d. Aalst
Inheritance of Behavior, p.83

99/18 J.C.M. Baeten and T. Basten
Partial-Order Process Algebra (and its Relation to Petri Nets), p. 79

99/19 J.C.M. Baeten and C.A. Middelburg
Real Time Process Algebra with Time-dependent Conditions, p.33.

99/20 Proceedings Conference Information 1999
Centrum voor Wiskunde en Informatica
12 November 1999, p.98

00/01 J.C.M. Baeten and J.A. Bergstra
Mode Transfer in process Algebra, p. 14

00/02 J.C.M. Baeten
Process Algebra with Explicit Termination, p. 17.

00/03 S. Mauw and M.A. Reniers
A process algebra for interworkings, p. 63.

00/04 R. Rloo, J. Hoornan and E. de Jong
Semantical Aspects of an Architecture for Distributed Embedded Systems*, p. 47.

00/05 J.F. Groote and M.A. Reniers
Algebraic Process Verification, p. 65.

00/06 J.F. Groote and J. v. Wamle
The Parallel Composition of Uniform Processes wit Data, p. 19