SOS Formats and Meta-Theory

20 Years After

MohammadReza Mousavi [1] Michel A. Reniers [2]
Jan Friso Groote [3]

Department of Computer Science, Eindhoven University of Technology (TU/e), P.O. Box 513, NL-5600 MB Eindhoven, The Netherlands

Abstract

In 1981 Structural Operational Semantics (SOS) was introduced as a systematic way to define operational semantics of programming languages by a set of rules of a certain shape [113]. Subsequently, the format of SOS rules became the object of study. Using so-called Transition System Specifications (TSS's) several authors syntactically restricted the format of rules and showed several useful properties about the semantics induced by any TSS adhering to the format. This has resulted in a line of research proposing several syntactical rule formats and associated meta-theorems. Properties that are guaranteed by such rule formats range from well-definedness of the operational semantics and compositionality of behavioral equivalences to security-, time- and probability-related issues. In this paper, we provide an overview of SOS rule formats and meta-theorems formulated around them.

Key words: Formal Semantics, Structural Operational Semantics, Rule Formats, Framework.

1 Introduction

Structural Operational Semantics [114,115,68] has become the common way to define operational semantics. Operational semantics defines the possible

[1] Corresponding Author. Also affiliated with Department of Electrical Engineering, Eindhoven University of Technology. Email: `M.R.Mousavi@tue.nl`.
This paper appeared in: TCS, 373:238-272, 2007. This preprint is made available for personal and academic purposes by a written permission from Elsevier Science.
[2] Email: `M.A.Reniers@tue.nl`
[3] Email: `J.F.Groote@tue.nl`

transitions that a piece of syntax can make during its “execution”. Each transition may be *labelled* by a message to be communicated to the outside world. Transitions of a composed piece of syntax can usually be defined in a generic way, in terms of the transitions of its constituent parts. This forms the central idea behind Structural Operational Semantics (SOS) (see [2,128] for formal links between SOS and the denotational approach to semantics).

Soon after its introduction, and due to its popularity, SOS itself became a subject of study, thus resulting in “SOS meta-theory” and “SOS rule formats”. The first SOS meta-result dates back to the Ph.D. Thesis of Robert de Simone [41] (in French), an excerpt of which appeared in 1985 [42]. Thus, the SOS 2005 workshop marks the 20th anniversary of SOS meta-theory.

Transition System Specifications (TSS’s), as introduced by Groote and Vaandrager in [65], are a formalization of SOS. By imposing syntactic restrictions on TSS’s one can deduce several interesting properties about their induced operational semantics. These properties range from well-definedness of the operational semantics [64,25,60] to security- [126,127] and probability-related issues [18,78]. The syntactic restrictions imposed by these meta-theorems usually suggest particular forms of deduction rules to be safe for a particular purpose and hence these meta-theorems usually define what is called a *rule format.*

In [5], an excellent overview is provided for existing rule formats up to the date of publication (2001). Since then, more formats have been proposed and we felt that in order to keep track, the field of SOS rule formats requires a fresh survey. Thus, to some extent, our work is complementary to [5] and by no means makes it obsolete. The goal of this paper somewhat differs from that of [5]; the present paper is most suitable for users of rule formats to find the appropriate results while [5] serves as a perfect introduction for researchers who would like to start developing new results in this field.

Therefore, we attempt to present an overview of all SOS rule formats defined in the literature, together with the meta-theorems formulated around them. The results are also put in a lattice to easily locate the most suitable format for a certain application. To do this, we define the concept of a TSS, in a far more general setting than [65], including the concepts of multi-sorted signatures and variable binding, inspired by the definition of [51]. This general definition of TSS can serve as a unifying framework and paves the way for studying semantic meta-theorems for SOS and comparing their underlying frameworks.

The rest of this paper is organized as follows. In Section 2, we define the notion of Transition System Specification and discuss its semantics. In Section 3 the hierarchy of formats is given and different syntactic features of SOS rules are studied. In Sections 5-14, we review rule formats and semantic meta-theorems about different SOS frameworks, as follows:

Section 5:	(Pre-)Congruence for behavioral equivalences (pre-orders);
Section 6:	Operational and equational conservativity of language extensions;
Section 7:	Generating equational theories and rewrite systems from the operational semantics specifications;
Section 8	Frameworks resulting from defining an ordering on rules and their associated meta-results;
Section 9:	Time-related properties;
Section 10:	Probability-related properties;
Section 11:	Expressiveness of operational semantics frameworks;
Section 12:	Compositional reasoning techniques for formalisms with an operational semantics;
Section 13:	Implementations of SOS (or closely related) instances / environments;
Section 14:	Other (more isolated) meta-results; results that stem from few papers, including: non-interference (security-related) and bounded non-determinism meta-theorems.

Acknowledgements. Peter Mosses, Simone Tini, Irek Ulidowski and anonymous referees provided useful comments on this article.

2 Transition System Specifications: Syntax and Semantics

2.1 Syntax

A TSS is supposed to define a number of transition relations and predicates on pieces of syntax. Thus, in order to define a TSS, it is essential to introduce syntax first. In practice, syntax is usually defined formally using a grammar.

Example 1 *The following is an example of a grammar for a simple process calculus:*

$$Proc := 0 \mid Act.Proc \mid Proc + Proc \mid Proc \,||\, Proc \mid \mu X.Proc \mid X,$$

where 0 *is a deadlocking process,* . *denotes action prefixing,* $+$ *is the nondeter-*

ministic choice operator, $||$ is the parallel composition operator and $\mu X.Proc$ denotes a recursive definition. In the above grammar, there are three syntactic classes: actions (Act), processes (Proc) and recursion variables (X).

Algebraic structures are convenient ways of modeling syntax. In order to define an algebraic structure for such a language, we have to fix a number of composition operators which take (possibly zero) parameters of certain sorts and compose them into a new piece of syntax. We also refer to composition operators as *function symbols* and the number and sorts of arguments taken by them is called their *arity*. The collection of function symbols with their arities is called a *signature*. In order to define the semantics structurally and generically, one needs to talk about open terms or terms with holes. The role of holes in terms is played by variables (sometimes called meta-variables or formal variables).

The following definition defines a general theory for the above concepts. Afterwards, we specify the syntax of the signature of the above process calculus to illustrate the formal definitions.

Definition 2 (Signatures, Terms and Substitutions) *A signature Σ consists of the following data:*

(1) A collection $\mathcal{S}$ of sorts represented by $S, S_1, S_2, \ldots$;
(2) A collection $\mathcal{N}$ of names such that for each sort S there exists an infinite set $\mathcal{N}_S$ of names, denoted by $n_S, m_S, \ldots$. We usually drop the sort indices of names as they are clear from the context.
(3) A collection of function symbols represented by $f, g, \ldots$ together with their arities which are of the form $\overrightarrow{S_1}.S'_1, \ldots, \overrightarrow{S_{ar(f)}}.S'_{ar(f)} \rightarrow S$, where $\overrightarrow{S_i}$ is a (possibly empty) list of sorts of the form $S_{i_1}, \ldots, S_{i_{k_i}}$ for some natural number k_i, and $ar(f)$ is a natural number denoting the number of parameters of function symbol f.

We fix a collection V of variables such that for each sort S there exists an infinite set V_S of variables, denoted by $x_S, y_S, \ldots$. We usually drop the sort indices of variables as they are clear from the context. The collection of terms of sort S, notation $\mathcal{T}_S$, is defined inductively by

- *a variable $x_S \in V_S$ is a term of sort S;*
- *a name $n_S \in \mathcal{N}_S$ is a term of sort S;*
- *for a function symbol f in Σ with arity $\overrightarrow{S_1}.S'_1, \ldots, \overrightarrow{S_{ar(f)}}.S'_{ar(f)} \rightarrow S$ and for all $1 \leq i \leq ar(f)$, if names $\overrightarrow{n_i}$ are of sorts S_i, respectively, and t_i are of sorts S'_i, then $f(\overrightarrow{n_1}.t_1, \ldots, \overrightarrow{n_{ar(f)}}.t_{ar(f)})$ is a term of sort S;*
- *for t a term of sort S, n_S a name of sort S', and t' a term of sort S', $t[t'/n_S]$ is a term of sort S. The part $\_[t'/n_S]$ is called the substitution harness.*

Concepts of free and bound names are defined as usual. A term is closed if no free variables occur in it. The set of all closed terms of sort S is denoted by $\mathcal{C}_S$. Terms are considered important up to α-conversion (i.e., renaming of bound names) and this is usually dealt with implicitly at the meta-level. The set of variables appearing in a term t is denoted by $vars(t)$.

A substitution σ is a collection of sort preserving functions σ_S from variables of sort S to terms of the same sort. A substitution that only maps variables to closed terms is called a closed *substitution. Application of a (closed) substitution is lifted to terms, formulas and sets of these as expected.*

In case we consider a single-sorted signature Σ with sort S, usually the sort and signature are identified and the notations V_Σ, $\mathcal{T}_\Sigma$, and $\mathcal{C}_\Sigma$ are used for V_S, $\mathcal{T}_S$, and $\mathcal{C}_S$, respectively.

In most of the remainder, unless stated otherwise, explicit substitutions are omitted from the term structure as they cause several difficulties when obtaining semantic meta results (the only exception is in Section 6, starting from Definition 36, where an operational conservativity result is given in the presence of explicit substitutions).

Example 3 *Considering the process calculus of Example 1, its signature has only two sorts A and P for actions and processes. The function symbols of this signature and their arities are as follows:*

$$a, \ldots \rightarrow A \qquad \textit{(Basic action constants)}$$

$$\_.\_ \quad A \times P \rightarrow P \qquad \textit{(Action prefixing)}$$

$$\_+\_ \quad P \times P \rightarrow P \quad \textit{(Nondeterminstic choice)}$$

$$\_\,||\,\_ \quad P \times P \rightarrow P \qquad \textit{(Parallel composition)}$$

$$\mu\_.\_ \quad P.P \rightarrow P \qquad \textit{(Recursion)}$$

Definition 4 (Predicate and Transition Formulae) *Fix sets Rel and Pred of relation and predicate symbols with fixed arities. Suppose relation symbol $r \in R$ is of relation arity $S_1 \times \ldots S_n \times S_{n+1}$ $(n \geq 1)$. We write $t_1 \overset{t_2,\ldots,t_n}{\longrightarrow}_r t_{n+1}$, where, for $1 \leq i \leq n+1$, t_i is of sort S_i, and call it a positive transition formula. We write $t_1 \overset{t_2,\ldots,t_n}{\nrightarrow}_r$ for a negative transition formula. In both cases $t_2, \cdots, t_n$ is called the label of the transition formula. A transition formula is* *a positive or negative transition formula.*

Suppose that predicate symbol $p \in Pred$ *is of predicate arity* $S_1 \times \ldots S_n$ $(n \geq 1)$*. We write* $(t_1 \ldots, t_{n-1}) \downarrow_p t_n$*, where* t_i *is of sort* S_i *and call it a positive predicate formula. We write* $(t_1 \ldots, t_{n-1}) \not\downarrow_p t_n$ *for a negative predicate formula. In both cases* $t_1, \cdots, t_{n-1}$ *is called the label of the predicate formula. A predicate formula is a positive or negative transition formula. A (positive or negative) formula is a (positive or negative) predicate or transition formula.*

Definition 5 (Transition System Specification(TSS)) *A TSS denoted by the pair* (Σ, D) *is a set of deduction rules* $d \in D$ *defined on a signature* Σ *where each deduction rule is of the following shape*[4] *:*

$$\frac{H}{C}$$

where H *is a set of formulae and* C *is a positive formula.*

For TSS's with constant labels, a distinguished sort L is dedicated to represent the labels. In the literature, as well as in the remainder of this paper, for TSS's with closed terms as labels, sort L is taken out of the signature Σ and such TSS's are then represented by a triple (Σ, L, D). In the rest of this paper, we use $l, l', l_i, \ldots$ to denote labels (in general).

Unless explicitly stated otherwise, in the rest of this paper, it is assumed that there is only one relation symbol and only one predicate symbol. These are conveniently denoted by $\rightarrow$ and $\downarrow$, respectively.

2.2 Semantics

For positive TSS's, i.e., TSS's without negative premises, the semantics of a TSS is clear; the transition relation (and the predicate) induced by a TSS is precisely the set of closed transitions (and predicate) formulae provable using the deduction rules. However, if there are negative premises in the semantical rules it is not self-evident anymore whether the rules define a transition relation in an unambiguous way. For example, consider the following two TSS's:

$$\frac{c \overset{a}{\nrightarrow}}{c \overset{a}{\rightarrow} c} \quad \Bigg| \quad \frac{c \overset{a}{\nrightarrow}}{c \overset{b}{\rightarrow} c} \qquad \frac{c \overset{b}{\nrightarrow}}{c \overset{a}{\rightarrow} c}$$

The left-hand-side TSS is rather paradoxical and the right-hand-side one is ambiguous in that it is not clear why one would prefer one of the two possible

[4] Note that the relation and predicate symbols are defined implicitly by their occurrence in the deduction rules.

transitions $c \xrightarrow{a} c$ or $c \xrightarrow{b} c$ over the other ([60] gives an overview of this issue, presents 11 different semantic interpretations of TSS's, and compares them formally).

In [64], the first criterion is given using which a TSS in the ntyft/ntyxt format (see Section 5) is guaranteed to have a well-defined semantics. This criterion, defined below for TSS's in general, is called (strict) stratification and is originally due to [55] in logic programming. It is an important property of a format when it guarantees that every set of rules unequivocally defines a transition relation. Below we give a more general definition of stratification for arbitrary TSS's.

Definition 6 (Stratification) *A stratification of a TSS with signature Σ is a function $\mathcal{S}$ from closed positive formulae to an ordinal such that for all deduction rules $\frac{H}{C}$ in the TSS, for all closed substitutions σ, and for all $h \in H$*

- $\mathcal{S}(\sigma(h)) \leq \mathcal{S}(\sigma(C))$ *if h is a positive formula;*
- $\mathcal{S}(\sigma(t \xrightarrow{l} t')) < \mathcal{S}(\sigma(C))$ *for all $t' \in \mathcal{C}_\Sigma$, if h is a negative transition formula $t \overset{l}{\nrightarrow}$;*
- $\mathcal{S}(\sigma((l) \downarrow t)) < \mathcal{S}(\sigma(C))$ *if h is a negative predicate formula $(l) \not\downarrow t$.*

A TSS is called stratified *when there exists a stratification function for it. If the measure decreases also from the conclusion to the positive premises, then the stratification is called strict.*

Later, in [25], the notions of semantics and well-definedness for TSS's were generalized to the notions of three-valued stable models and complete TSS's. In order to define these notions, we need a few auxiliary notions for provability and truth.

Definition 7 (Provability and Truth) *A deduction rule $\frac{\Phi}{\phi}$ is* provable *from a TSS tss, denoted by tss $\vdash \frac{\Phi}{\phi}$, when there exists a well-founded*[5] *upwardly branching tree with formulae as nodes and of which*

- *the root is labelled by ϕ;*
- *if a node is labelled by ψ and the nodes above it form the set K then one of the following two cases hold:*
  - · $\psi \in \Phi$ *and $K = \emptyset$;*
  - · ψ *is a positive transition formula and $\frac{K}{\psi}$ is an instance of a deduction*

[5] A tree is well-founded if and only if each node of the tree having a finite distance to the root node.

rule in tss (a deduction rule r with a substitution σ applied to it).

A negative transition formula $\phi = p \overset{l}{\nrightarrow}$ is true *for a set PF of positive formulae, denoted by $PF \vDash \phi$ when there exists no p' such that $p \xrightarrow{l} p' \in PF$. A negative predicate formula $\phi = (l) \not\downarrow p$ is true for a set PF of positive formulae, denoted by $PF \vDash \phi$ when $(l) \downarrow p \notin PF$. Formulae $p \xrightarrow{l} p'$ and $(l) \downarrow p$ respectively deny $p \overset{l}{\nrightarrow}$ and $(l) \not\downarrow p$ and vice versa. A set NF of negative formulae is true for the set PF, denoted by $PF \vDash NF$ when for all $\phi \in NF$, $PF \vDash \phi$.*

As said before, for a positive TSS's, the notion of provability defines the semantics of TSS's in a straightforward manner by requiring that a closed positive formula is in the semantic model of TSS if and only if it is provable from the TSS from an empty set of premises. However, for TSS's with negative premises the issue is more complicated and for such TSS's the following notion from [116,25] defines a general semantics.

Definition 8 (Three-Valued Stable Models) *A pair (C, U) of disjoint sets of* positive closed formulae *is called a three-valued stable model for TSS tss (where C stands for* Certain *and U for* Unknown*; the third value is determined by the formulae not in $C \cup U$) when*

- *for all $\phi \in C$, $tss \vdash \dfrac{N}{\phi}$ for a set N of negative closed transition formulae such that $C \cup U \vDash N$;*
- *for all $\phi \in C \cup U$, $tss \vdash \dfrac{N}{\phi}$ for a set N of negative closed transition formulae such that $C \vDash N$.*

In [116,60], it has been shown that every TSS admits a least three-valued stable model with respect to the information theoretic ordering (i.e., $(C, U) \leq (C', U')$ when $C \subseteq C'$ and $U' \subseteq U$). A TSS is called complete [60] (or positive after reduction [25]) if for its least three-valued stable model (C, U), $U = \emptyset$. A stratified TSS is indeed complete but not all complete TSS's can be stratified [25].

The proof theoretic counterpart for the notion of three-valued stable model is the following notion of well-supported proof.

Definition 9 (Well-Supported Proof) *A well-supported proof in a TSS tss for a formula ϕ is a well-founded upwardly branching tree with formulae as nodes and of which*

- *the root is labeled by ϕ;*
- *if a node is labeled by a positive formula ψ and the nodes above it form the*

set K then $\frac{K}{\psi}$ is an instance of a deduction rule in tss (a deduction rule r with a substitution σ applied to it);

- *if a node is labeled by a negative formula ψ and the nodes above it form the set K then for all provable deduction rules $\frac{N}{\gamma}$ such that N is a set of negative formulae and γ denies ψ, a formula in N denies one in K.*

Note that the notion of well-supported proof is consistent in that no two formulae denying each other can be given a well-supported proof. A TSS is complete (in the model theoretic sense) if and only if for all formulae ϕ, either ϕ itself or a formula denying it can be given a well-supported proof.

2.3 Complementary Frameworks

Sometimes deduction rules are furnished with complementary constructs such as an ordering relation on deduction rules or an equational specification. In this section we review such complementary frameworks.

Ordering the Deduction Rules. One way to avoid the use of negative premises (and sometimes predicates) is by defining an order among deduction rules. Then, a deduction rule of a lower order may be applied to prove a formula only when there is no deduction rule with a higher order applicable. For example, the semantics of the priority operator [7] can be expressed in terms of a number of rules of the following form

$$(r_a)\frac{x \xrightarrow{a} x'}{\theta(x) \xrightarrow{a} \theta(x')}$$

with an ordering $\ll$ defined by $r_a \ll r_b$ whenever $a < b$. The semantics of the sequential composition operator can also be defined by the following rules

$$(r_x)\frac{x \xrightarrow{l} x'}{x; y \xrightarrow{l} x'; y} \qquad (r_y)\frac{y \xrightarrow{l} y'}{x; y \xrightarrow{l} y'}$$

with the ordering $\ll$ defined by $r_y \ll r_x$. This way, the second argument of sequential composition can take over, only when the first argument cannot make a transition, i.e., has terminated (we do not consider unsuccessful termination or deadlock in this simple setting). The implications of introducing an order among deduction rules and its possible practical use are investigated in [133–135,111].

Equational Specifications. Structural congruences are equational addenda to SOS specification which can define inherent properties of function symbols or define some function symbols in terms of the others. For example, the following equation specifies that the order of arguments in a parallel composition does not matter or in other words, that parallel composition is commutative.

$$x \,||\, y \equiv y \,||\, x$$

In [100], the addition of equational specifications to SOS specifications is studied in detail.

3 A hierarchy of rule formats

Since the introduction of the first rule format in [42], some 20 other ones have been added. In order to keep track of them we made an overview of the existing rule formats in Figure 1. The lattice presented there has SOS frameworks as nodes, ordered by syntactic inclusion of rule formats (mainly based on the syntactic features). The most general format can be found at the top and more specific formats at lower positions. The arrows indicate syntactical inclusion of rule formats [6]. The one inclusion that is not syntactic but possibly requires some translation of syntactic constructs is indicated by a dotted line.

A node

Format Name

Syntactic Features

Semantic Meta-Theorems [references]

in this lattice represents a format with additional information on the type of syntactic features that are included and the semantic meta-theorems that are given for this format in the literature (with references). Consider the following example taken from the lattice:

NTyft (NTree)

SS, CT, LA, NWF,

CPY, NEG, INF.

C($\underline{\leftrightarrow}_s$) *[64,48]*

It describes the NTyft (Ntree) format which is defined for single sorted TSS's (SS) with closed terms as labels (CT) and allows for lookahead (LA), non-well-

[6] As a consequence we tend to neglect finiteness assumptions about the number of rules, the number of function symbols in signatures and the like.

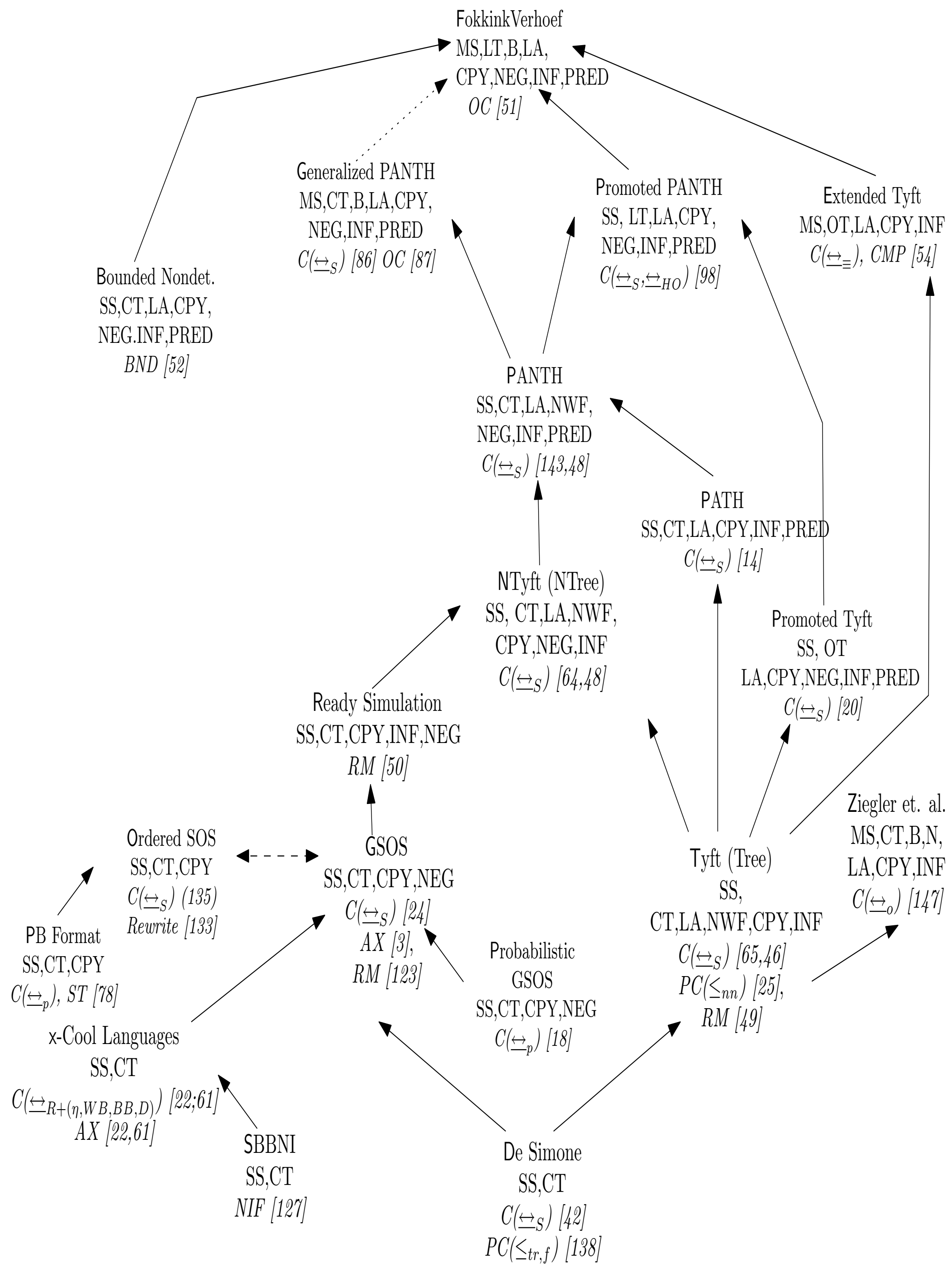

Fig. 1. A hierarchy of existing SOS formats

founded rules (NWF), copying (CPY), negative premises (NEG), and infinite premises (INF). For this format, $C(\underline{\leftrightarrow}_s)$ *[48,64]* indicates that a congruence meta-theorem for strong bisimilarity is presented in [48,64].

Table 1 lists abbreviations used for indicating the syntactic features of the rule formats that appear in the hierarchy of rule formats from Figure 1. In

Syntactic Features	**Abbreviations**
Many-Sortedness of the Signature	
Single-Sorted	SS
N-Sorted	NS
Many-Sorted	MS
Labels	
Closed Terms	CT
Open Terms	OT
Lists of Open Terms	LT
Binders and Names	
Binders	B
Names	N
Lookahead	LA
Non-well-founded Rules	NWF
Copying Variables	CPY
Negative Premises	NEG
Infinite Premises	INF
Predicates	PRED

Table 1
Syntactic features of rule formats (cf. Fig. 1)

Section 4, these syntactic features are introduced and illustrated. Table 2 lists abbreviations used for indicating the semantic meta-results in the hierarchy. These are further explained from Section 5 onwards.

4 Syntactic Features of Rule Formats

The syntactic features that are encountered in the hierarchy are described in the rest of this section. The abbreviations can be found in Table 1.

Many-Sortedness of the Signature Based on the number of sorts allowed in the signature, an SOS framework may be classified in the following three categories:

SS Single-sorted TSS's: This is the most common framework in the literature.

Semantic Meta-Theorems

Semantic Meta-Theorems	**Abbreviations**
Congruence for x-Bisimulation	C($\underline{\leftrightarrow}_x$)
Congruence for x-equality	C($\approx_x$)
Pre-congruence for x-pre-order	PC($\leq_x$)
Operational Conservativity	OC
Deriving Sound and Complete Axiomatization	AX
Comparison of Induced Equality Classes	CMP
Reasoning Methods	RM
Non Interference (Security-related [119])	NIF
Bounded Non-determinism	BND
Stochasticity	ST

Table 2
Short-hands for theorems used in Figure 1

It has a single sort for operational states which is usually called the sort of *processes* (and terms from this sort are process terms). In this framework, there is usually a sort for constant labels, as well.

NS N-Sorted TSS's: A framework may only allow for a fixed number of sorts participating in the signature. An example of such frameworks is the process-tyft format of [104] where there are two distinguished sorts of processes and data. Apart from these two sorts that are used to define the states of the semantics, there is a sort for constant labels.

MS Multi-Sorted TSS's: In such frameworks, there is no restriction on the sorts allowed for constructing terms.

The TSS of [54] has a special status with respect to its allowed signatures. Namely, it requires a special sort for processes and at least one (necessarily different) sort for labels. Furthermore, it requires that process sorts should not participate in function symbols with label sorts as targets.

Labels Labels are terms that may appear as parameters of transition relations and predicates in the deduction rules. SOS frameworks can be classified with respect to the kind of labels they afford as follows.

CT Closed terms as labels: Many SOS frameworks assume a special sort for labels and only allow for closed terms (alternatively, constants) of this sort to appear as labels. Such SOS frameworks thus forbid any correlation between valuation of terms in the source and the targets of transitions, on one hand,

and the labels, on the other hand, through the use of common variables. Most SOS framewroks reviewed in this paper only allow for constants as labels.

OT Open terms as labels: Frameworks defined in [51,54,20,98] allow for arbitrary open terms as labels.

Open terms are used as labels in a number of cases in transition system specifications. Higher-order process calculi [27,125,121] are examples of formalisms exploiting this feature.

Two other frameworks that use labels with some structure on them are the Modular SOS framework of [96,97] and the Enhanced Operational Semantics of [44]. The former assumes that labels are arrows of a category and thus comes equipped with composition and identity. The latter approach codes the derivation tree of transitions on their labels.

LT Lists of Terms as Labels: Most SOS frameworks only allow for a single term as label. The only existing exceptions are those of [51,98].

Names and Binders In many contemporary process algebras and calculi, concepts of names, (actual and formal) variables and name abstraction (binding) are present and even serve as a basic ingredient. For example, in the π-calculus [91–93], names are first-class citizens and the whole calculus is built around the notion of passing names among concurrent agents. Less central, yet important, instances of these concepts appear in different process algebras in the form of the recursion operator, the infinite sum operator and the time-integration operator (cf., for example, [91], [82,117] and [13], respectively). Hence, it is interesting to accommodate the concept of names in the TSS framework.

B The first step towards specifying the above-mentioned constructs is to allow for binding signatures. There have been a few attempts in this direction. Proposals for SOS frameworks with binding signatures are formulated in [147,51,86,87,72,120,146].

N Another related issue in this regard, is the concept of fresh names and freshness contexts. In the semantics and notions of behaviorial equivalence for some formalisms (such as the π-calculus), one encounters “side-conditions” about freshness of names (or statements such as “name n does not appear free in term t”). In order to capture these notions, some capacities must be foreseen. The nominal techniques of Gabbay and Pitts [53] and the $FO\lambda^{\Delta\nabla}$ formalism of Miller and Tiu [88] provide good possibilities for incorporating these notions in the SOS meta-theory and the first attempt in this direction has been made by [147]. [45] adopts an alternative model of names and freshness contexts to extend the de Simone format with such concepts.

Apart from the above mentioned formats, all other rule formats we mention in the remainder of this paper do not support names and binders.

Lookahead A framework allows for lookahead if a deduction rule in the framework may have two premises with a variable in the target of one of the premises being present in the source of the other. An example of a deduction rule with lookahead is the following.

$$\frac{x \xrightarrow{\tau} y \quad y \xrightarrow{l} z}{x \xrightarrow{l} z}$$

The above rule from [56] is used to combine silent (τ) and ordinary transitions in order to implement a weak semantics (by ignoring silent steps) inside a strong semantic framework.

Non-well-founded Rules The variable dependency graph of a deduction rule is a graph of which the nodes are variables and there is an edge between two variables if one appears in the source or label and the other in the target of (the same) positive premise in the deduction rule. A deduction rule is well-founded when all the backward chains of variables in the variable dependency graph are finite. A TSS is well-founded when all its deduction rules are.

All practical instances of SOS specifications are well-founded. Well-foundedness also comes very handy in the proof of semantic meta-results for SOS frameworks. It is of theoretical interest whether a framework allows for non-well-founded deduction rules or not.

Copying A framework has the copying feature if it allows for repetition of variables in different premises and in the target of the conclusion and sharing a variable between the source of a premise and target of the conclusion (called branching premises, explicit copying and implicit copying, respectively in [130]). A simple example of explicit copying is the second rule in the following TSS which defines the semantics of the `while` construct.

$$\frac{\neg \mathit{Hold}[b, M]}{\langle \texttt{while}\ (b)\ \texttt{do}\ P\ \texttt{od}, M\rangle \downarrow}$$

$$\frac{\mathit{Hold}[b, M]}{\langle \texttt{while}\ (b)\ \texttt{do}\ P\ \texttt{od}, M\rangle \rightarrow \langle P; \texttt{while}\ (b)\ \texttt{do}\ P\ \texttt{od}, M\rangle}$$

Negative Premises Negative premises are a complicating factor in SOS frameworks. They cause several complications with respect to the semantics of TSS's which are rather difficult to solve. In other words, it is not immediately clear what can be considered a "proof" for a negative formula.

To our knowledge, in practice, the first example of negative premises in SOS appeared in [11,7] in the specification of the semantics of the following priority operator $\theta(\_)$.

$$\frac{x \xrightarrow{a} x' \quad \forall_{b>a}\ x \stackrel{b}{\nrightarrow}}{\theta(x) \xrightarrow{a} \theta(x')}$$

The above deduction rule states that a parameter of $\theta(\_)$ can perform a transition with label a if no transition with a label b of higher priority can be performed (according to a given partial ordering $>$).

Infinite Premises It is an interesting theoretical question whether a framework allows for infinite number of premises or not. Also practically, when dealing with infinite domains (e.g., infinite basic actions, data or time domains), it is sometimes useful to have deduction rules with infinite premises. The above deduction rule for the priority operator may have infinite premises if there is a chain of priorities with infinitely many (different) basic actions.

The following example from [107] illustrates another possible use of deduction rules with infinite premises:

$$\frac{x \xrightarrow{a} y}{\partial_H(x) \xrightarrow{a} \partial_H(x')}\ [a \notin H] \qquad \frac{\forall_{a \in A \setminus H}\ x \stackrel{a}{\nrightarrow}}{\partial_H(x) \xrightarrow{\chi} \delta}$$

The above deduction rules define the semantics of the encapsulation operator $\partial_H(\_)$ which forbids its parameter from performing transitions in H. If the parameter cannot perform any ordinary action allowed by ∂_H then it makes a transition to the deadlocking process δ. If the set of basic actions is infinite, then for some finite H, the deduction rule in the right-hand side has infinite (negative) premises.

Predicates Predicates are useful syntactic features which are used to specify phenomena such as termination or divergence. The following deduction rules define the concept of termination $\downarrow \_$ for a simple signature containing a constant 1 representing the empty process and a binary non-deterministic choice $+$ [12]:

$$\frac{}{\downarrow 1} \qquad \frac{\downarrow x}{\downarrow x + y} \qquad \frac{\downarrow y}{\downarrow x + y}$$

5 (Pre-)Congruence for Behavioral Equivalences

5.1 Preliminary Definitions

Given an operational semantics, it is interesting to observe when two systems show the same behavior. It is also interesting to check whether a particular system is a restricted implementation of the other. To check these, we have to have notions of *behavioral equivalence* and *behavioral pre-order*, respectively. It is very much desired for a notion of behavioral equivalence (pre-order) to be compositional or in technical terms to be a *congruence* (pre-congruence). There is a myriad of notions of behavioral equivalence and pre-order in the concurrency literature [57,59]. Correspondingly, there are a number of rule formats guaranteeing these notions to be (pre-)congruences [65,23,22,61]. In the remainder, we confine ourselves to single-sorted frameworks. In such frameworks, the arity of a function symbol can be conveniently expressed by a natural number (representing the number of parameters on the left-hand side of the arrow). The only congruence meta-theorems for multi-sorted frameworks are those of [54,86,51,147]. The only congruence results accommodating open terms as labels are [54,51,98].

We start with defining the notion of congruence for relations on closed terms.

Definition 10 ((Pre-)Congruence) *An equivalence (pre-order) $R \subseteq \mathcal{C}_\Sigma \times \mathcal{C}_\Sigma$ is a (pre-)congruence with respect to a signature Σ if and only if for all function symbols f from Σ and all $\overrightarrow{p}, \overrightarrow{q} \in \mathcal{C}_\Sigma$, if $\overrightarrow{p}\ R\ \overrightarrow{q}$ then $f(\overrightarrow{p})\ R\ f(\overrightarrow{q})$.*

5.2 Congruence of strong bisimilarity

The first congruence formats were defined for the notion of strong bisimilarity, defined below.

Definition 11 (Bisimulation and Bisimilarity [110]) *Given a transition relation $\rightarrow\ \subseteq \mathcal{C}_\Sigma \times L \times \mathcal{C}_\Sigma$ and a predicate $\downarrow \subseteq \mathcal{C}_\Sigma$, a symmetric relation $R \subseteq \mathcal{C}_\Sigma \times \mathcal{C}_\Sigma$ is a bisimulation relation if and only if, for all $p, q \in \mathcal{C}_\Sigma$ such that $(p, q) \in R$, it satisfies*

- *for all $p' \in \mathcal{C}_\Sigma$ and $l \in L$, if $p \xrightarrow{l} p'$, then $q \xrightarrow{l} q'$ for some $q' \in \mathcal{C}_\Sigma$ such that $(p', q') \in R$; then $p \xrightarrow{l} p'$ for some $p' \in \mathcal{C}_\Sigma$ such that $(p', q') \in R$;*
- *for all $l \in L$, $(l) \downarrow p$ if and only if $(l) \downarrow q$.*

Two closed terms p and q are bisimilar, $p \leftrightarrow q$, if and only if there exists a bisimulation relation R such that $(p, q) \in R$. Two closed terms p and q

are bisimilar with respect to a transition system specification tss, denoted by $tss \vdash p \leftrightarrow q$, if and only if they are bisimilar with respect to the semantics of tss.

There are good reasons for considering strong bisimilarity as an important notion of behavioral equivalence. Here, we mention a few.

(1) Strong bisimilarity usually gives rise to elegant and neat theories and it turns out that congruence formats for it are also much more elegant and compact than those for other (weaker) notions;
(2) For finite state processes, strong bisimilarity can be checked very efficiently in practice [108] while some weaker notions are intractable [75];
(3) Other (weaker) notions of behavioral equality can often be coded in strong bisimilarity [56].

So, it is not surprising that the first standard congruence format was geared toward strong bisimilarity. This format, which uses the positive framework with constant labels, was proposed by de Simone in [42].

Definition 12 (de Simone Format) *A deduction rule is in the* de Simone *format if and only if it has the following form:*

$$\frac{\{x_i \xrightarrow{l_i} y_i \mid i \in I\}}{f(\overrightarrow{x}) \xrightarrow{l} t} \quad [Cond(\overrightarrow{l_i}, l)].$$

where x_i and y_i are distinct variables, f is a function symbol from the signature, I is a subset of the set $\{1, \ldots, ar(f)\}$ (indices of arguments of f), and t is a process term that only contains variables among y_i's for $i \in I$ and x_j's for $j \notin I$ and does not have repeated occurrences of variables, l_i's and l are constant labels and $Cond$ is a condition on the labels of the premises and the label of the conclusion.

A TSS is in the de Simone *format if all its deduction rules are.*

The side conditions in the above rule format are usually left out since such rules can be replaced by the set of all rules with concrete labels satisfying the side condition.

Bloom, Istrail and Meyer, in their study of the relationship between bisimilarity and completed-trace congruence [24], define an extension of the de Simone format, called GSOS (for Structural Operational Semantics with Guarded recursion), to capture *reasonable* language definitions. The GSOS rule format extends the de Simone rule format by allowing for copying and negative premises.[7] The GSOS format is formally defined as follows.

Definition 13 (GSOS Format) *A deduction rule is in the GSOS format if and only if it has the following form:*

$$\frac{\{x_i \xrightarrow{l_{ij}} y_{ij} \mid i \in I, 1 \leq j \leq m_i\} \cup \{x_j \overset{l'_{jk}}{\nrightarrow} \mid j \in J, 1 \leq k \leq n_j\}}{f(\overrightarrow{x}) \xrightarrow{l} t}$$

where f is a function symbol, x_i ($1 \leq i \leq ar(f)$) and y_{ij}'s ($i \in I$ and $1 \leq j \leq m_i$) are all distinct variables, I and J are subsets of $\{1, \ldots, ar(f)\}$, m_i and n_j are natural numbers (to set an upper bound on the number of premises), $vars(t) \subseteq \{x_i \mid 1 \leq i \leq ar(f)\} \cup \{y_{ij} \mid i \in I, 1 \leq j \leq m_i\}$ and l_{ij}'s, l'_{jk}'s and l are constant labels. A TSS is in the GSOS format when it has a finite signature, a finite set of labels, a finite set of deduction rules and all its deduction rules are in the GSOS format.

It should be noted that in the original formulation of the GSOS format from [24], the TSS is required to have unary operators $a.\_$ and binary operator $+$ with their standard deduction rules

$$\frac{}{a.x_1 \xrightarrow{a} x_1} \qquad \frac{x_1 \xrightarrow{a} x'_1}{x_1 + x_2 \xrightarrow{a} x'_1} \qquad \frac{x_2 \xrightarrow{a} x'_2}{x_1 + x_2 \xrightarrow{a} x'_1}.$$

In later publications such as [5], this requirement has been omitted.

Another orthogonal extension of the de Simone format is called the tyft/tyxt format[8] and is first formulated in [65]. This format allows for lookahead, copying and an infinite set of premises.

Definition 14 (Tyft/tyxt Format [65]) *A rule is in the tyft format if and only if it has the following form:*

$$\frac{\{t_i \xrightarrow{l_i} y_i \mid i \in I\}}{f(\overrightarrow{x}) \xrightarrow{l} t}$$

where x_i and y_i are all distinct variables (i.e., for all $i, i' \in I$ and $1 \leq j, j' \leq ar(f)$, $y_i \neq x_j$ and if $i \neq i'$ then $y_i \neq y_{i'}$ and if $j \neq j'$ then $x_j \neq x_{j'}$), f is

[7] Note that this does not mean that a TSS in the de Simone format is also in the GSOS format since the GSOS format has additional finiteness assumptions.

[8] Tyft/tyxt is a code representing the structure of symbols in the deduction rules, namely, a general term (t) in the source of the premises, a variable (y) in the target of the premises, a function symbol (f) or a variable (x) in the source of the conclusion and a term (t) in the target of the conclusion.

a function symbol from the signature, I is a (possibly infinite) set of indices, the t_i's and t are arbitrary terms and the l_i's and l are constant labels.

A rule is in tyxt *format if it is of the above form but the source of the conclusion is a variable distinct from all variables that appear in the targets of the premises. A TSS is in the* tyft *format when all its deduction rules are. A TSS is in the* tyft/tyxt *format when all its deduction rules are either in the* tyft *or in the* tyxt *format.*

Any TSS in the tyft/tyxt format can be reduced to an equivalent TSS (inducing the same transition relations) in the tyft format. In [65], to prove congruence of strong bisimilarity for TSS's in the tyft/tyxt format, well-foundedness of the TSS is assumed. Later, in [46], it is shown that the well-foundedness constraint can be relaxed and that for every non-well-founded TSS in the tyft/tyxt format, a TSS exists that induces the same transition relation and is indeed well-founded.

The merits of the two extensions were merged in [64] where negative premises were added to the tyft/tyxt format, resulting in the ntyft/ntyxt format.

Definition 15 (Ntyft/ntyxt Format [64]) *A rule is in the* ntyft *format if and only if it has the following form:*

$$\frac{\{t_i \xrightarrow{l_i} y_i \mid i \in I\} \cup \{t_j \overset{l'_j}{\nrightarrow} \mid j \in J\}}{f(\overrightarrow{x}) \xrightarrow{l} t}$$

The same conditions as of the tyft *format hold for the positive premises and the conclusion. The negative premises have constant labels. There is no particular constraint on the terms appearing in the negative premises. Set J is the (possibly infinite) set of indices of negative premises. A rule is in the* ntyxt *format if it is of the above form but the source of conclusion is a variable distinct from all targets of premises. A TSS is in the* ntyft *format when all its deduction rules are. A TSS is in the* ntyft/ntyxt *format when all its deduction rules are either in the* ntyft *or in the* ntyxt *format.*

As explained before, introduction of negative premises in the ntyft/ntyxt format brings about doubts regarding the well-definedness of the semantics. In [25], it has been shown that for complete TSS's in the well-founded ntyft/ntyxt format strong bisimilarity is a congruence. The well-foundedness assumption was also used in [64] and was shown to be redundant in [48].

Finally, the path format [14] for Predicates And Tyft/tyxt Hybrid format) and the panth format [144] (for Predicates And Negative Tyft/tyxt Hybrid format) extend the tyft/tyxt and the ntyft/ntyxt with predicates, respectively.

Definition 16 (Panth Format [144]) *A rule is in the* panth *format if and only if it has the following form:*

$$\frac{\{(l_i) \downarrow t_i \;\; or \; t_i \xrightarrow{l_i} y_i \mid i \in I\} \cup \{(l'_j) \not\downarrow t_j \;\; or \; t_j \stackrel{l'_j}{\nrightarrow} \mid j \in J\}}{(l) \downarrow t \;\; or \; f(\overrightarrow{x}) \xrightarrow{l} t \;\; or \; x \xrightarrow{l} t}$$

The same conditions as for the ntyft/ntyxt *format hold for the premises and the conclusion. A TSS is in the* panth *(*path*) format when all its deduction rules are (and it contains no negative premises).*

Theorem 17 (Congruence of Strong Bisimilarity for Panth [144]) *For a complete TSS in the* panth *format, strong bisimilarity is a congruence.*

The promoted tyft/tyxt format [20] extends the tyft/tyxt format with open terms as labels. In [98], the authors present a format that simplifies the promoted tyft/tyxt format by eliminating some of the restrictions on the labels of the transition relation, and adding to it predicates and negative premises and the use of lists of terms as labels of transition relations. It is therefore also a generalization of the panth format. The following definitions and theorem are formulated in the setting with multiple transition relations and predicates and lists of open terms as labels.

An operator is called volatile w.r.t. a certain relation or predicate if it occurs as a label of such a relation or predicate in a premise of a deduction rule with a variable that occurs in the source or in the right-hand side of a positive premise of that same deduction rule.

Definition 18 (Volatile Operators) *An operator f is* volatile *for relation r (or predicate p) when there exists a deduction rule of the form*

$$\frac{\{(l_i) \downarrow_{p_i} t_i \;\; or \; t_i \xrightarrow{l_i}_{r_i} t'_i \mid i \in I\} \cup \{(l_j) \not\downarrow_{p_j} t_j \;\; or \; t_j \stackrel{l_j}{\nrightarrow}_{r_j} \mid j \in J\}}{(l) \downarrow_{p'} \; t \;\; or \; t \xrightarrow{l}_{r'} t'}$$

where $f(\overrightarrow{t})$ is a subterm of a component of l_m for some $m \in I \cup J$ such that $r = r_m$ ($p = p_m$) and $vars(\overrightarrow{t}) \cap vars(t) \neq \emptyset$ or $vars(\overrightarrow{t}) \cap vars(t'_i) \neq \emptyset$ for some $i \in I$.

Definition 19 (Promoted Panth Format [98]) *A rule is in the* promoted panth *format if and only if it has the following form:*

$$\frac{\{(l_i) \downarrow_{p_i} \; t_i \;\; or \; t_i \xrightarrow{l_i}_{r_i} y_i \mid i \in I\} \cup \{(l_j) \not\downarrow_{p_j} \; t_j \;\; or \; t_j \stackrel{l_j}{\nrightarrow}_{r_j} \mid j \in J\}}{(l) \downarrow_p f(\overrightarrow{x}) \;\; or \; f(\overrightarrow{x}) \xrightarrow{l}_r t}$$

where I and J are disjoint and the following criteria are satisfied:

(1) all the variables x_i $(1 \leq i \leq ar(f))$ and y_i $(i \in I)$ and the variables in l are pairwise distinct;
(2) if a component of l_k $(k \in I \cup J)$ is a variable then it is not among the x_i's and y_i's;
(3) for all components t of l:
  (a) if t contains a volatile g for r (for p) then t is of the form $g(\overrightarrow{z})$ where all z_i's are distinct variables and for all $k \in I \cup J$, all components of l_k containing a variable among $\overrightarrow{z}$ are of the form $g'(\overrightarrow{t})$ where g' is volatile for r_k (for p_k);
  (b) if there is a volatile operator for r (for p) in the signature and if t is a variable z then for all $k \in I \cup J$, all components of l_k containing z are either z itself or are of the form $g'(\overrightarrow{t})$ where g' is volatile for r_k (for p_k).

A TSS is in the promoted panth *format when all its deduction rules are.*

Theorem 20 (Congruence of Strong Bisimilarity for Promoted Panth) *For a stratified and well-founded TSS in* promoted panth *format, strong bisimilarity is a congruence.*

We expect that in the above theorem the assumption about stratification can be replaced by the more general completeness assumption. In [103], the authors show that the well-foundedness assumption cannot be dropped in the above theorem.

In [86], the panth format is extended for multi-sorted signatures and variable binding. The resulting format is called generalized panth. This covers the problem of operators such as recursion or choice over a time domain. The notion of bisimilarity used differs from the traditional one in the sense that closedness under α-conversion and closedness w.r.t. substitution for bound variables are required (see [86, Definition 4.1]). The issue of binding operators for multi-sorted process terms is also briefly introduced in [5].

5.3 Congruence of weak equivalences

For the congruence of weaker notions of bisimilarity a number of standard formats have been proposed. A major class of such formats are the Cool languages formats introduced in [22] which prove congruence of (rooted) weak and branching bisimilarity. Recently, in [61], these formats have been reformulated and additional formats have been given to prove congruence of (rooted) delay and (rooted) η-bisimilarity.

Before we give the formats, we first define the notions of equivalence. Note that the formats from [22] and [61] are for a TSS without predicates. Therefore, we have given the definitions of the weak equivalences for a TSS without predicates. In the following definitions it is assumed that $\tau \in L$. The notations $\Rightarrow$ and $\xrightarrow{(l)}$ from [5] are used with their usual meanings, i.e., $p \Rightarrow p'$ indicates an arbitrary sequence of consecutive τ-transitions: $p \equiv p_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} p_n \equiv p'$ $(n \geq 1)$, and $p \xrightarrow{(l)} p'$ abbreviates $p \xrightarrow{l} p'$ or $l = \tau \wedge p = p'$.

Definition 21 (Weak Bisimilarity) *A symmetric relation $R \subseteq \mathcal{C}_\Sigma \times \mathcal{C}_\Sigma$ is a weak bisimulation relation if and only if, for all $p, q \in \mathcal{C}_\Sigma$ such that $(p, q) \in R$, it satisfies*

- *for all $p' \in \mathcal{C}_\Sigma$ and $l \in L$, if $p \xrightarrow{l} p'$, then $q \Rightarrow q_1 \xrightarrow{(l)} q_2 \Rightarrow q'$ for some $q_1, q_2, q' \in \mathcal{C}_\Sigma$ such that $(p', q') \in R$.*

Two closed terms p and q are weakly bisimilar, $p \underline{\leftrightarrow}_w q$ if and only if there exists a weak bisimulation relation R such that $(p, q) \in R$. Two closed terms p and q are rooted *weakly bisimilar, $p \underline{\leftrightarrow}_{rw} q$, if the following conditions are satisfied:*

- *whenever $p \xrightarrow{l} p'$ there exist q_1, q_2 and q' such that $q \Rightarrow q_1 \xrightarrow{l} q_2 \Rightarrow q'$ and $p' \underline{\leftrightarrow}_w q'$;*
- *whenever $q \xrightarrow{l} q'$ there exist p_1, p_2 and p' such that $p \Rightarrow p_1 \xrightarrow{l} p_2 \Rightarrow p'$ and $p' \underline{\leftrightarrow}_w q'$.*

Definition 22 (Branching Bisimilarity) *A symmetric relation $R \subseteq \mathcal{C}_\Sigma \times \mathcal{C}_\Sigma$ is a branching bisimulation relation if and only if, for all $p, q \in \mathcal{C}_\Sigma$ such that $(p, q) \in R$, it satisfies*

- *for all $p' \in \mathcal{C}_\Sigma$ and $l \in L$, if $p \xrightarrow{l} p'$, then $q \Rightarrow q_1 \xrightarrow{(l)} q'$ for some $q_1, q' \in \mathcal{C}_\Sigma$ such that $(p, q_1) \in R$ and $(p', q') \in R$.*

Two closed terms p and q are branching bisimilar if and only if there exists a branching bisimulation relation R such that $(p, q) \in R$. Two closed terms p and q are rooted *branching bisimilar, $p \underline{\leftrightarrow}_{rb} q$, if the following conditions are satisfied:*

- *whenever $p \xrightarrow{l} p'$ there exists q' such that $q \xrightarrow{l} q'$ and $p' \underline{\leftrightarrow}_b q'$;*
- *whenever $q \xrightarrow{l} q'$ there exists p' such that $p \xrightarrow{l} p'$ and $p' \underline{\leftrightarrow}_b q'$.*

Definition 23 (η-Bisimilarity) *A symmetric relation $R \subseteq \mathcal{C}_\Sigma \times \mathcal{C}_\Sigma$ is an η-bisimulation relation if and only if, for all $p, q \in \mathcal{C}_\Sigma$ such that $(p, q) \in R$, it satisfies*

- *for all $p' \in \mathcal{C}_\Sigma$ and $l \in L$, if $p \xrightarrow{l} p'$, then $q \Rightarrow q_1 \xrightarrow{(l)} q_2 \Rightarrow q'$ for some $q_1, q_2, q' \in \mathcal{C}_\Sigma$ such that $(p, q_1) \in R$ and $(p', q') \in R$.*

Two closed terms p and q are η-bisimilar, $p \underline{\leftrightarrow}_\eta q$, if and only if there exists an

η-bisimulation relation R such that $(p,q) \in R$. Two closed terms p and q are rooted *η-bisimilar, $p \underline{\leftrightarrow}_{r\eta} q$, if the following conditions are satisfied:*

- *whenever $p \xrightarrow{l} p'$ there exist q_1 and q' such that $q \xrightarrow{l} q_1 \Rightarrow q'$ and $p' \underline{\leftrightarrow}_{\eta} q'$;*
- *whenever $q \xrightarrow{l} q'$ there exist p_1 and p' such that $p \xrightarrow{l} p_1 \Rightarrow p'$ and $p' \underline{\leftrightarrow}_{\eta} q'$.*

Definition 24 (Delay Bisimilarity) *A symmetric relation $R \subseteq \mathcal{C}_\Sigma \times \mathcal{C}_\Sigma$ is a delay bisimulation relation if and only if, for all $p, q \in \mathcal{C}_\Sigma$ such that $(p,q) \in R$, it satisfies*

- *for all $p' \in \mathcal{C}_\Sigma$ and $l \in L$, if $p \xrightarrow{l} p'$, then $q \Rightarrow q_1 \xrightarrow{(l)} q'$ for some $q_1, q' \in \mathcal{C}_\Sigma$ such that $(p', q') \in R$.*

Two closed terms p and q are delay bisimilar, $p \underline{\leftrightarrow}_d q$, if and only if there exists a delay bisimulation relation R such that $(p,q) \in R$. Two closed terms p and q are rooted *delay bisimilar, $p \underline{\leftrightarrow}_{rd} q$, if the following conditions are satisfied:*

- *whenever $p \xrightarrow{l} p'$ there exist q_1 and q' such that $q \Rightarrow q_1 \xrightarrow{l} q'$ and $p' \underline{\leftrightarrow}_d q'$;*
- *whenever $q \xrightarrow{l} q'$ there exist p_1 and p' such that $p \Rightarrow p_1 \xrightarrow{l} p'$ and $p' \underline{\leftrightarrow}_d q'$.*

Definition 25 *Consider a TSS in the* positive *GSOS format, i.e., all premises of all deduction rules are positive. For an operator f, the* rules of f *are the rules with source $f(\overrightarrow{x})$.*

An operator is straight *if it has no rules in which a variable occurs multiple times in the left-hand sides of its premises. An operator is* smooth *if moreover it has no rules in which a variable occurs both in the target and the left-hand side of a premise.*

An argument i ($1 \le i \le n$) of an operator f is active *if f has a rule in which x_i appears as the left-hand side of a premise.*

A variable x occurring in a term t is receiving in t *if t is the target of a rule in which x is the right-hand side of a premise. An argument i of an operator f is* receiving *if a variable x is receiving in a term t that has a subterm $f(t_1, \ldots, t_n)$ with x occurring in t_i.*

Definition 26 (Patience Rule) *A rule of the form*

$$\frac{x_i \xrightarrow{\tau} y}{f(x_1, \ldots, x_n) \xrightarrow{\tau} f(x_1, \ldots, x_{i-1}, y, x_{i+1}, \cdots, x_n)}$$

with $1 \le i \le n$ is called a patience *rule for the i^{th} argument of f.*

Definition 27 *A TSS in GSOS format is* two-tiered *if its operations can* *be partitioned into abbreviations and principal operators, and for every abbreviation f a principal operator f^* is specified, together with a substitution $\sigma_f : \{x_1, \ldots, x_{ar(f)-1}\} \rightarrow \{x_1, \ldots, x_{ar(f)-1}\}$, such that the rules of f are*

$$\left\{ \frac{\sigma_f(H)}{f(\overrightarrow{x}) \xrightarrow{l} \sigma_f(t)} \;\middle|\; \frac{H}{f^*(\overrightarrow{x}) \xrightarrow{l} t} \text{ is a rule of } f^* \right\}.$$

Definition 28 (Cool Formats) *A TSS is* WB cool *if it is in the positive* GSOS *format, is two-tiered, and (1) all principal operators are straight; (2) patience rules are the only rules of principal operators with τ-premises; (3) every active argument of a principal operator has a patience rule; (4) if argument i of f is receiving, then argument i of f^* has a patience rule; (5) all principal operators are smooth.*

The formats DB cool, HB cool *and* BB cool *are defined likewise, but skipping clause (4) for* DB cool *and* BB cool *and clause (5) for* HB cool *and* BB cool.

For the case that there are only principal operators, the WB cool and BB cool formats coincide with the ones of [22], whereas the DB cool format coincides with the eb format of [135].

Theorem 29 (Congruence for Weak Notions of Bisimilarity) *If a TSS is in* WB cool *format, then $\underline{\leftrightarrow}_w$ is a congruence. If a TSS is in* DB cool *format, then $\underline{\leftrightarrow}_d$ is a congruence. If a TSS is in* HB cool *format, then $\underline{\leftrightarrow}_\eta$ is a congruence. If a TSS is in* BB cool *format, then $\underline{\leftrightarrow}_b$ is a congruence.*

Definition 30 (Ruloid) *For transition rule r, let $RHS(r)$ denote the set of the right-hand sides of the premises of that rule. For a given TSS in positive* GSOS *format, the class of ruloids is the smallest set that satisfies: (1) $\dfrac{x \xrightarrow{l} y}{x \xrightarrow{l} y}$ is a ruloid, for every $x, y \in V$ and $l \in L$; (2) if σ is a substitution, the given tss has a rule $\dfrac{H}{s \xrightarrow{l} t}$ for some H, s, t, l, and for every premise $x \xrightarrow{l'} y$ in H there is a ruloid $r_y = \dfrac{H_y}{\sigma(x) \xrightarrow{l'} \sigma(y)}$ such that the sets $RHS(r_y)$ are pairwise disjoint and each $RHS(r_y)$ is disjoint with $var(\sigma(s))$, then $\dfrac{\bigcup_{y \in H} H_y}{\sigma(s) \xrightarrow{l} \sigma(t)}$ is a ruloid.*

Definition 31 (RWB Cool Format) *A TSS is in the* RWB cool *format if it is positive and the operators can be partitioned into tame and wild ones, such that (1) the target of every rule contains only tame operators; (2) the sub-TSS of tame operators is in the* WB cool *format; (3) for each rule $\dfrac{H}{s \xrightarrow{a} t}$ there is a term u and a substitution $\sigma : var(u) \rightarrow var(s)$ such that there is a ruloid*

$\dfrac{K}{u \xrightarrow{a} v}$ *with* $\sigma(K) = H$ *and* $\sigma(v) = t$*, and for every premise* $x \xrightarrow{c} y$ *in* K*, there is a rule* $\dfrac{\sigma(x) \xrightarrow{\tau} y}{s \xrightarrow{\tau} \sigma(u[y/x])}$*; (4) if argument* i *of* f *is receiving, then argument* i *of* f^* *has a patience rule.*

Theorem 32 (Congruence for Rooted Weak Bisimilarity) *If a TSS is in RWB cool format, then* $\underline{\leftrightarrow}_{rw}$ *is a congruence.*

Fokkink in [47] presents the RBB safe format which induces congruence for rooted branching bisimilarity and generalizes the RB cool format of [22] to the setting with negative premises and predicates. Fokkink admits that this generalisation is complicated and that no examples are found in the literature that are RB cool with abbreviations.

5.4 Other Congruence and Precongruence Formats

In this section, we discuss some formats for congruence and precongruence from the literature. In [132], Ulidowski shows that the τDeS format from [130] results in precongruence of the testing preorder from [106].

Definition 33 (τDeS Format) *For a given deduction rule in the de Simone format of the form*

$$\frac{\{x_i \xrightarrow{l_i} x_i' \mid i \in I\}}{f(\vec{x}) \xrightarrow{l} t}$$

with $l_i \neq \tau$ *(for* $i \in I$*), the associated patience rules are the deduction rules of the form*

$$\frac{x_i \xrightarrow{\tau} x_i'}{f(\vec{x}) \xrightarrow{\tau} f(x_1, \ldots, x_{i-1}, x_i', x_{i+1}, \ldots, x_{ar(f)-1})}$$

one for each $i \in I$.

A TSS is in the τDeS format when it has a finite signature, a finite set of labels, a finite set of deduction rules and if its deduction rules consist of de Simone deduction rules of the above form and their associated patience rules only.

Obviously any TSS in τDeS format is also in the de Simone format.

Theorem 34 (Congruence for the τDeS Format [132]) *Testing preorder is a precongruence for any TSS in the τDeS format.*

The τDeS format is a subformat of the GSOS format and therefore strong bisimilarity is a congruence for any such TSS. Also, the τDeS format is a subformat of the ISOS format from [130]. Therefore, it gives congruence for refusal simulation preorder [130] and eager bisimilarity [134].

Klin in [77] restricts the GSOS format to the CTr-format which guarantees congruence for completed trace equivalence.

In [58], the *ready simulation* format is proposed that induces congruence for ready simulation. This format is the ntyft/ntyxt format without the lookahead feature. The ready simulation format is further restricted in [23] to obtain pre-congruence for readiness, ready traces and failures pre-orders. Note that pre-congruence for a pre-order implies congruence for the corresponding equivalence (the kernel of the pre-order).

In [98], the higher-order panth format is presented which induces congruence for higher-order bisimilarity [6].

The rule format of [147] guarantees that open-bisimilarity [122] is a congruence. In [45], a rule format is presented which guarantees quasi-open bisimilarity to be a congruence.

In [39], tile bisimilarity is defined and syntactic criteria for guaranteeing its congruence are presented.

In his seminal paper [72], Howe proposes a general method for proving pre-congruence of applicative similarity (and thus congruence of applicative bisimilarity [1]). Furthermore, [72] presents a rule format for pre-congruence of applicative similarity. Along the same lines, the GDSOS format of [120], guarantees the existence and uniqueness of least fixed points for recursive semantic definitions.

6 Conservativity of Language Extensions

Operational semantics of languages may be extended by adding new pieces of syntax to the signature and new rules to the set of deduction rules. A number of meta-theorems have been proposed to check whether extensions do not change the behavior of the old language and whether they preserve equalities among old terms. To extend a language defined by a TSS, one may have to combine an existing signature with a new one. However, not all signatures can be combined into one as the arities of the function symbols may clash. To prevent this, we define two signatures to be *consistent* when they agree on the arity of the shared function symbols. In the remainder, we always assume that extended and extending TSS's are consistent. The following definition formalizes the concept of operational extension.

Definition 35 (Extension of a TSS) *Consider TSS's $tss_1 = (\Sigma_1, D_1)$ and $tss_2 = (\Sigma_2, D_2)$ with consistent signatures. The extension of tss_1 with tss_2, denoted by $tss_1 \oplus tss_2$, is defined as $(\Sigma_1 \cup \Sigma_2,\ D_1 \cup D_2)$.*

Traditionally, an extension of a TSS is called operationally conservative w.r.t. the original one when no new transitions and predicates can be derived for old terms. In [65], Groote and Vaandrager present the first meta-theorem for establishing operational conservativity. It applies to TSS's that are in the tyft/tyxt format and for which the old deduction rules are pure (and thus well-founded) and the new deduction rules must contain a new operator in the source of the conclusion. Subsequently, Groote [64] extends this result to TSS's with negative premises (ntyft/ntyxt), and Bol and Groote [25] show that the restriction to the ntyft/ntyxt format can be omitted as long as the TSS is complete (positive after reduction). Verhoef, in [143], broadens the applicability of the meta-theory by allowing new deduction rules with an old term as a source under certain conditions.

More general instances of meta-theorems for operational conservativity are formulated in [51,87,145]. In the rest of this section, we review the results of [51], which gives the most detailed account of this issue. The work in [51] extends the previously mentioned works with many-sorted signatures and general binding mechanisms.

In [51], it is recognized that there are several ways of giving meaning to a TSS with negative premises. Fokkink and Verhoef present a definition of operational conservativity, given below, that is stronger than the traditional definition in the sense that it implies the traditional notion of conservative extension for several such interpretations including the three-valued stable model interpretation (Definition 8).

Definition 36 (Operational Conservativity) *Given two TSS's tss_1 and tss_2, $tss_1 \oplus tss_2$ is an operationally conservative extension of tss_1 if for each derived deduction rule $\frac{N}{C}$ of $tss_1 \oplus tss_2$ with N a set of negative closed formulae and C a positive closed formula with a source from the signature of tss_1, we have that it is also a derived deduction rule of tss_1.*

Definition 37 (Fresh Terms) *Let Σ_1 and Σ_2 be signatures. Freshness of a term over signature Σ_2 w.r.t. signature Σ_1 is defined inductively as follows: (1) $f(\vec{n_1}.t_1, \ldots, \vec{n_{ar(f)}}.t_{ar(f)})$ is fresh if f is in Σ_2 but not in Σ_1 or t_i is fresh for some $1 \le i \le ar(f)$, (2) $t[t'/n]$ is fresh if t is fresh.*

Next, we formulate sufficient conditions to prove operational conservativity. But before that, we need a few auxiliary definitions. The notion of source-dependency replaces the notions pure and well-founded as used by [65].

Definition 38 (Source-Dependency) *For a deduction rule d, and a collection of sorts $\mathcal{S}$, the source-dependent variables modulo $\mathcal{S}$ in d are defined inductively as follows:*

- *All variables appearing outside the substitution harness in the source of the conclusion of a deduction rule are source-dependent modulo $\mathcal{S}$ in d;*
- *All variables of sort S for some $S \in \mathcal{S}$ are source-dependent modulo $\mathcal{S}$ in d;*
- *All variables that occur in the label or the target of a positive premise of deduction rule d outside a substitution harness are source-dependent modulo $\mathcal{S}$ in d if all variables in the source of the premise are source-dependent modulo $\mathcal{S}$ in d.*

Definition 39 (Reduced Rules) *For a deduction rule $d = (H, c)$, the reduced rule with respect to a signature Σ is defined by $\rho(d, \Sigma) \doteq (H', c)$ where H' is the set of all premises from H which have a Σ-term as a source.*

Theorem 40 (Operational Conservativity Meta-Theorem [51]) *Given two TSS's $tss_1 = (\Sigma_1, D_1)$ and $tss_2 = (\Sigma_2, D_2)$, $tss_1 \oplus tss_2$* [9] *is an operationally conservative extension of tss_1 if:*

(1) $\mathcal{S}$ is a collection of sorts such that for each $S \in \mathcal{S}$ there are no fresh terms of sort S from $\mathcal{T}_{\Sigma_1 \cup \Sigma_2}$ w.r.t. Σ_1;
(2) for all $d_1 \in D_1$, all variables that occur in d_1 are source-dependent modulo $\mathcal{S}$ in d_1;
(3) for all $d_2 \in D_2$ at least one of the following holds:
  (a) the source of the conclusion is fresh, or
  (b) $\rho(d, \Sigma_1)$ has a source-dependent positive premise $t \xrightarrow{l} t'$ or $(l) \downarrow t$ such that $t \in \mathcal{T}_{\Sigma_1}$, all variables that occur in t are source-dependent modulo $\mathcal{S}$ in $\rho(d, \Sigma_1)$, and one of the terms appearing in l is fresh.

In [101], a slightly more liberal notion of operational conservativity, called *orthogonality*, is introduced and a meta-theorem is proposed for it. Orthogonality allows for the addition of new transitions and predicates on the old syntax provided that these new transitions and predicates do not change the behavioral equalities among old terms.

7 Generating Equational Theories

Equational theories are central notions to process algebras [8,19,71,90]. They capture the basic intuition behind the algebra, and the models of the algebra

[9] Note that although the notation $\oplus$ has been defined for TSS's with a designated sort for labels (i.e., a TSS in triple notation), it is possible to define $\oplus$ on more general TSS's (thus represented by a tuple) in a meaningful way as well.

are expected to respect this intuition (e.g., the models induced by operational semantics modulo bisimilarity). One of the benefits of having equational theories is that they enable reasoning at the level of syntax without committing to particular models of the semantics. When the semantic model of behavior (e.g., the transition system associated to a term) is infinite, these techniques may come in very handy.

To establish a reasonable link between the operational model and the equational theory of the algebra, a notion of behavioral equality should be fixed. Ideally, the notion of behavioral equivalence should coincide with the closed derivations of the equational theory. One side of this coincidence is captured by the *soundness* theorem which states that all closed derivations of the equational theory are indeed valid with respect to the particular notion of behavioral equality. The other side of the coincidence, called *completeness*, states that all induced behavioral equalities are derivable from the equational theory, as well. These concepts are formalized in what follows.

Definition 41 (Equational Theory) *An* equational theory *or* axiomatization (Σ, E) *is a set of equalities* E *on a signature* Σ *of the form* $t = t'$*, where* $t, t' \in \mathcal{T}$*. A closed instance* $p = p'$*, for some* $p, p' \in \mathcal{C}_\Sigma$*, is derivable from* E*, denoted by* $E \vdash p = p'$*, if and only if it is in the smallest congruence relation on closed terms induced by the equalities of* E*.*

An equational theory (Σ, E) *is* sound *with respect to a TSS tss (also on signature* Σ*) and a particular notion of behavioral equality* $\sim$ *if and only if for all* $p, p' \in \mathcal{C}_\Sigma$*, if* $E \vdash p = p'$*, then it holds that tss* $\vdash p \sim p'$*. It is* complete *if the implication holds in the other direction.*

In [3], an automatic method for generating sound and complete equational theories from GSOS specifications is presented. It is assumed that there are only transition relations $\rightarrow\ \subseteq \mathcal{C}_\Sigma \times L \times \mathcal{C}_\Sigma$.

Definition 42 (Disjoint Extension) *Consider TSS's* $tss_1 = (\Sigma_1, L_1, D_1)$ *and* $tss_2 = (\Sigma_2, L_2, D_2)$*. TSS* tss_2 *is a* disjoint extension *of a TSS* tss_1 *if and only if* $\Sigma_1 \subseteq \Sigma_2$ *and* $D_1 \subseteq D_2$ *and the operators from* Σ_1 *do not occur in the sources of the deduction rules from* $D_2 \setminus D_1$*.*

Next, we define when a TSS does not allow infinite traces. Such a TSS is called *trace-finite.*[10] In [3], also syntactic criteria are given for establishing trace-finiteness of a TSS.

Definition 43 (Trace-Finiteness) *Let tss be a TSS in the* GSOS *format.*[11]

[10] In [3], trace-finiteness is called well-foundedness. We have used this term already to name another property in Section 4.

[11] In [3], this notion is defined for TSS's w.r.t. the supported model semantics. As

A term $p \in \mathcal{C}_\Sigma$ is trace-finite *iff there exists no infinite sequence $p_1, l_1, p_2, l_2, \ldots$ of closed terms p_i and labels l_i such that $p \equiv p_1$ and, for all $i \geq 1$, $p_i \xrightarrow{l_i} p_{i+1}$. The TSS tss is* trace-finite *iff all terms in $\mathcal{C}_\Sigma$ are well-founded.*

Theorem 44 *Let tss be a trace-finite TSS in the* GSOS *format. Then, there are a disjoint extension tss' of tss and a finite equational theory T' such that T' is a sound and complete axiomatization of bisimilarity on closed terms from tss'.*

A generalization of this result to even non-trace-finite TSS's is also presented in [3], though this requires the addition of the *Approximation Induction Priciple* (AIP) from [10] to the equational theory.

Theorem 45 *Let tss be a trace-finite TSS in the* GSOS *format. Then, there are a disjoint extension tss' of tss and a finite equational theory T' such that T' extended with AIP is a sound and complete axiomatization of bisimilarity on closed terms from tss'.*

The techniques from [3] were extended in [15] to cater for explicit termination of processes. This approach, although more complicated in nature, gives rise to more intuitive and more compact sets of equations compared to the original approach of [3]. The resulting format is called the tagh format. The name tagh format denotes *termination and* GSOS *hybrid* format. It is assumed that there are only transition relations $\rightarrow\ \subseteq \mathcal{C}_\Sigma \times L \times \mathcal{C}_\Sigma$ and predicates $\downarrow \subseteq \mathcal{C}_\Sigma$.

Definition 46 (Tagh Format) *A deduction rule is in the* tagh *format if and only if it has one of the following forms:*

(1) a tagh*-transition rule*

$$\frac{\{x_i \xrightarrow{l_{ip}} y_{ip} \mid i \in I, p \in P_i\} \cup \{x_j \overset{l'}{\nrightarrow} \mid j \in J, l' \in B_j\} \cup \{\downarrow x_k \mid k \in K\}}{f(\vec{x}) \xrightarrow{l} t}$$

where $I, J, K \subseteq \{1, \cdots, ar(f)\}$; for all $i \in I$, P_i is a non-empty finite index set; for all $j \in J$, $B_j \subseteq L$; for $1 \leq m \leq ar(f)$, $i \in I$, and $p \in P_i$, x_m and y_{ip} are pairwise distinct variables; $vars(t) \subseteq \{x_1 \ldots, x_{ar(f)}\} \cup \{y_{ip} \mid i \in I, p \in P_i\}$;

(2) a tagh*-termination rule*

$$\frac{\{\downarrow x_k \mid k \in K\}}{\downarrow f(\vec{x})}$$

where $x_1, \ldots, x_{ar(f)}$ are pairwise distinct variables and $K \subseteq \{1, \ldots, ar(f)\}$.

we only use the term for TSS's in the GSOS format, there is no misunderstanding about the meaning of a TSS regardless the chosen semantics.

A TSS is in the tagh *format when it has a finite signature, a finite set of labels, a finite set of deduction rules and all its deduction rules are in the* tagh *format.*

The above definition of the tagh format is different from the original definition from [15] in the sense that there it is assumed that certain operators and deduction rules are present in the TSS. We changed the definition to facilitate comparison of the following theorem to the results from [3] and [4]. It is obviously the case that the tagh format extends the GSOS format and is subsumed by the panth format.

Theorem 47 *Let tss be a TSS in the* tagh *format. Then, there are a disjoint extension tss' of tss and a finite equational theory T' such that T' extended with AIP is a sound and complete axiomatization of bisimilarity on closed terms from tss'.*

The GSOS and the tagh formats only allow the generation of an axiomatization in case the signature, the set of action labels, and the set of deduction rules are finite. In [4], Aceto defines the infinitary GSOS format. It extends the GSOS format by allowing for a countable signature, a countable set of action labels, and a countable set of deduction rules. Then, the sub-format regular GSOS allows for generation of a complete and sound axiomatization, containing the *Recursive Specification Principle* [16] of bisimilarity. The restrictions that are covered by this regular GSOS format only allow for the specification of processes with a finite labelled transition system.

Bloom [22] has shown that the approach of [3] can also be used for generating axioms for $\underline{\leftrightarrow}_{rb}$, and [61] claims that this is also the case for $\underline{\leftrightarrow}_{r\eta}$. In the latter work, also an adaptation of the approach of [3] is presented that yields finite sound and complete axiomatizations for $\underline{\leftrightarrow}_{rb}$ and $\underline{\leftrightarrow}_{rd}$.

Axiom systems for pre-orders have been generated, too, see for instance [132]. Along the same lines, [133] generates prioritized rewrite systems for TSS's with an ordering on deduction rules (see Section 8). Here we present the axiomatization results of [132] for the τDeS format (see Definition 33).

Theorem 48 *Let tss be a trace-finite TSS in the τDeS format. Then, there are a disjoint extension tss' of tss and a finite equational theory T' such that T' is a sound and complete axiomatization of testing preorder on closed terms from tss'.*

In [131], a similar result is obtained for refusal simulation [130].

The condition of well-foundedness can be replaced by the weaker condition of τ-convergence in combination with the introduction of an induction principle à la AIP. In [105], a meta-theorem is developed to generate sound commutativity axioms from TSS's in the tyft/tyxt format.

8 Ordered SOS

Ordered SOS [135] replaces the need for negative premises by adding an ordering among rules. A formal definition of Ordered SOS is given below.

Definition 49 (Ordered TSS) *An Ordered TSS is a pair $(tss, <)$ where tss is a TSS containing only positive GSOS rules and $<$ is a class of (partial) orders $<_f$ for all function symbols f in the signature.*

As explained informally in Section 2.3, the idea of Ordered SOS is to assert the impossibility of certain transitions (thus, test negative premises) by attempting to apply initially rules higher in the ordering that contain these transitions in the premises. The following example illustrates this.

Example 50 *Consider the following deduction rules:*

$$(r)\frac{x \xrightarrow{a} y}{f(x) \xrightarrow{a} f(y)} \qquad (r')\frac{x \xrightarrow{b} y \quad x \stackrel{a}{\nrightarrow}}{f(x) \xrightarrow{a} f(y)}$$

The right-hand-side rule checks for impossibility of a-transitions on the only argument of f. In Ordered SOS, this check can be replaced by declaring $(r'') <_f (r)$, where deduction rule (r'') is obtained from deduction rule (r') by removing the negative premise. Then, by the semantics of Ordered SOS, it is guaranteed that the (r'') can only be applied when (r) cannot and thus, when the argument x cannot make an a-transition.

To realize this, the semantics of Ordered SOS is defined as follows.

Definition 51 *Consider an ordered TSS $((\Sigma, L, D), <)$. Let $size(p)$ be the size of closed term p which is defined to be $1 + max(\{size(p_i) \mid 1 \leq i \leq ar(f)\})$ for $p = f(\vec{p})$. Also for an f-rule $r \in D$, let $higher(r)$ be $\{r' \mid r <_f r'\}$. Then, the transition relation $\rightarrow$ induced by $(tss, <)$ is $\bigcup_{k<\omega} \rightarrow^k$, where $\rightarrow^k$ is defined below.*

$$\begin{aligned} p \xrightarrow{a} p' \in \rightarrow^k \textit{ when } size(p) = k \wedge \\ \exists_{r \in D, \sigma : V \rightarrow \mathcal{C}_\Sigma}\, \sigma(conc(r)) = p \xrightarrow{a} p' \wedge \\ \sigma(pre(r)) \subseteq \bigcup_{m<k} \rightarrow^m \wedge \\ \forall_{r' \in D}\; r' \in higher(r) \Rightarrow \sigma(pre(r')) \nsubseteq \bigcup_{m<k} \rightarrow^m \end{aligned}$$

Here $pre(r)$ and $conc(r)$ denote the set of premises and the conclusion of a deduction rule r, respectively.

It is shown in [135] that every transition system definable by a TSS in the GSOS format is definable by Ordered SOS and vice versa. The idea of the translation from GSOS to Ordered SOS is very similar to the idea of Example 50, that is, rules with negative premises are placed below rules that have a positive premise on the same argument. Such rules (with appropriate positive tests) are not always present and to remedy this, one has to add some auxiliary rules with such premises and set their order in such a way that these auxiliary rules can never be applied themselves but can indeed replace the negative tests for rules below them. For the converse direction, the GSOS translation of a rule r, is a set of rules (with the same conclusion as that of (r)), each of which have the premises of (r) together with a set of premises making sure that the rules above (r) are all not applicable.

In [135], it is shown that taking the ordering relation to be a partial order does not have any consequence for the expressiveness of definable Ordered SOS languages. In [99], a general approach to the semantics of ordered SOS is studied along the lines of [60].

9 Timed Properties

Timed extensions of programming and specification languages allow one to specify and reason about (quantitative) temporal properties of programs and specifications. The extension of an operational semantics with time is usually realized by adding a time transition relation to the transition system (see e.g., [94]) or sometimes by annotating the action transitions with timing information. In the remainder, we only consider the former approach for it is the only approach studied in the SOS meta-theory.

There are several design decisions involved in the process of extending a language with time (e.g., the dilemmas concerning absolute vs. relative time, instantaneous vs. durational actions and different time progress assumptions). One of the main decisions in such extensions concerns the choice of time domain. The simplest time domain is the discrete time domain which comes equipped with a total ordering, a zero and a next constructor (i.e., non-negative natural numbers). More involved time domains can be dense or partially ordered. In the remainder, we concentrate on the timed properties concerning timed extensions with a discrete (and totally ordered) time-domain. Initial results about an arbitrary time domains are given [76] but they are very complicated and very much geared towards a particular timed process algebra.

In order to extend a language with a discrete-time aspect, it suffices to add a new transition relation $\rightsquigarrow$ representing a unit-time delay (together with a number of time-related operators such as an operator for specifying deadlines).

In [136], a number of timed properties are studied in the context of ordered SOS languages with a notion of divergence-sensitive rooted delay bisimilarity, called rooted eager bisimilarity. The timed properties studied in [136] are listed below. In what follows, $p \rightsquigarrow$ ($p \xrightarrow{l}$) means that there exists a p' such that $p \rightsquigarrow p'$ ($p \xrightarrow{l} p'$), $p \Downarrow$ means that p converges, i.e., there is no infinite succession of τ-steps starting from p. Below, p, p' and p'' are closed terms and are all universally quantified.

(1) Time determinism: if $p \rightsquigarrow p'$ and $p \rightsquigarrow p''$ then $p' = p''$;
(2) Timelock freeness: $p \rightsquigarrow$;
(3) Weak timelock freeness: if $p \Downarrow$ then $p(\Rightarrow \circ \rightsquigarrow)$;
(4) Maximal (time) progress: if $p \xrightarrow{\tau}$ then $p \not\rightsquigarrow$;
(5) Patience (non-urgency): if $p \stackrel{\tau}{\not\rightarrow}$ then $p \rightsquigarrow$;
(6) Time persistence: if $p \xrightarrow{l}$ and $p \rightsquigarrow p'$ then $p' \xrightarrow{l}$, for any l.

Sufficient conditions for satisfying the above properties are given in [136]. To do this, first a set of syntactic criteria on the ordered operational rules are defined in order to guarantee that rooted eager bisimilarity is a congruence. Operators with rules satisfying the syntactic criteria are called rebo (for rooted eager bisimulation ordered) operators. The main feature of rebo operators is that τ-transitions only appear in patience rules (see Definition 25). Such rules have a special syntactic form and satisfy some constraints with respect to their ordering.

Then, an extension of a language with rebo operators into time settings is defined (timed rebo languages) which guarantees congruence of (timed) rooted early bisimilarity and time determinism. Such an extension adds appropriate discrete-time-transition rules which are in accordance with the definition and ordering of the patience rules in the original language. Subsequently, further restrictions are put on timed rebo languages which guarantee patience (weak timelock freeness), timelock freeness and time persistence.

In [76], a syntactic format is developed which induces congruence for strong (timed) bisimilarity and time-determinism for languages with a discrete notion of time. To do this, GSOS languages are taken as the basis and their timed extension is defined as the partitioning of the language into action and time description parts. Thus, rules with an action transition in the conclusion cannot have time transitions in their premises and vice versa. Then, in addition to GSOS constraints, the time-transition part should satisfy the syntactic criteria of the dsl format defined below.

Definition 52 (Dsl Format) *A rule of the following form is called a* dsl*-prerule (for "deterministic single label" rule).*

$$\frac{\{x_i \rightsquigarrow y_i \mid i \in I\} \cup \{x_j \not\rightsquigarrow \mid j \in J\}}{f(\vec{x}) \rightsquigarrow t}$$

where I and J are subsets of $\{k \mid 1 \leq k \leq ar(f)\}$, all y_i's (for all $i \in I$) and variables in $\vec{x}$ are pairwise distinct and $vars(t) \subseteq \{x_k, y_i \mid 1 \leq k \leq ar(f), i \in I\}$. A dsl*-prerule is consistent if $I \cap J = \emptyset$ and complete if $I \cup J = \{k \mid 1 \leq k \leq ar(f)\}$. A consistent and complete* dsl*-prerule is called a* dsl*-rule.*

A set of dsl*-rules is in* dsl *format when for all two f-rules of the following shapes:*

$$\frac{\{x_i \rightsquigarrow y_i \mid i \in I\} \cup \{x_j \not\rightsquigarrow \mid j \in J\}}{f(\vec{x}) \rightsquigarrow t} \qquad \frac{\{x_i \rightsquigarrow y_i \mid i \in I'\} \cup \{x_j \not\rightsquigarrow \mid j \in J'\}}{f(\vec{x}) \rightsquigarrow t'}$$

it holds that $(I \cap J') \cup (I' \cap J) \neq \emptyset$. Such f-rules are called mutually exclusive.

Theorem 53 *For a TSS in* GSOS *format where deduction rules with an action transition in the conclusion cannot have time transitions in their premises and deduction rules with a time transition in the conclusion are in the* dsl *format, time-transitions are deterministic and timed bisimilarity is a congruence.*

It is informally argued (and the arguments seem correct) that the consistency and completeness conditions can be relaxed. Thus, a set of mutually-exclusive dsl-prerules is sufficient for the purpose of Theorem 53.

10 Probability-Related Properties

There are several sorts of operational semantics for probabilistic systems which have different combinations of probability distribution and non-determinism (see [124,17] for an overview).

Two leading examples of probabilistic semantics are reactive and generative semantics. In the reactive semantics, the discrete probability distribution is defined for each label l of outgoing transitions while for the generative model, the discrete probability distribution is defined on all outgoing transitions (possibly with different labels). (A discrete probability distribution μ, or just a distribution, on set S is a function $\mu : S \rightarrow [0,1]$ such that $\sum_{s \in S} \mu(s) = 1$.)

For a closed term p, a reactive probabilistic transition with a label l and distribution μ on closed terms, is denoted by $p \xrightarrow{l} \rightsquigarrow \mu$. Intuitively, this means that term p can make an l transition to term q (in the domain of μ) with the probability $\mu(q)$. A generative transition from term p with distribution μ on the product of labels and terms (i.e., $L \times \mathcal{C}_\Sigma$) is denoted by $p \rightsquigarrow \mu$. For both types of systems, we write $p \xrightarrow{a[\rho]} q$ and by that, for reactive system, we mean $p \xrightarrow{a} \rightsquigarrow \mu$ and $\mu(q) = \rho$ and for generative system, we mean $p \rightsquigarrow \mu$ and $\mu(a, q) = \rho$. Notions of bisimilarity for reactive and generative systems are defined as follows [81,62,124].

Definition 54 (Reactive Probabilistic Bisimilarity [81]) *Let P and Q be arbitrary sets. Consider a relation $R \subseteq P \times Q$. Then the lifting of $R \subseteq (P \to [0,1]) \times (Q \to [0,1])$, denoted by $\equiv_R$, is defined as follows.*

For two distributions $\mu : P \to [0,1]$ and $\mu' : Q \to [0,1]$, $\mu \equiv_R \mu'$ when there exists a distribution $\nu : (P \times Q) \to [0,1]$ satisfying the following constraints.

- $\forall_{p \in P} \sum_{q \in Q} \nu(p,q) = \mu(p)$;
- $\forall_{q \in Q} \sum_{p \in P} \nu(p,q) = \mu'(q)$;
- $\forall_{p \in P, q \in Q} \nu(p,q) \neq 0 \Leftrightarrow (p,q) \in R$.

A symmetric relation $R \subseteq \mathcal{C}_\Sigma \times \mathcal{C}_\Sigma$ is a reactive bisimulation relation when for all $p, q \in \mathcal{C}_\Sigma$, if $p \xrightarrow{a} \rightsquigarrow \mu_p$ then there exists a μ_q such that $q \xrightarrow{a} \rightsquigarrow \mu_q$ and $\mu_p \equiv_R \mu_q$.

Definition 55 (Generative Probabilistic Bisimilarity [62]) *Consider an equivalence relation $R \subseteq P \times P$. The A-lifting of R is denoted by $\equiv_{A,R}$ and is defined as follows.*

For distributions $\mu, \mu' : (L \times P) \to [0,1]$, $\mu \equiv_{A,R} \mu'$ when $\forall_{a \in A} \forall_{p \in P} \sum_{q \in [p]_R} \mu(a,q) = \sum_{q \in [p]_R} \mu'(a,q)$.

A symmetric relation $R \subseteq \mathcal{C}_\Sigma \times \mathcal{C}_\Sigma$ is a generative bisimulation relation when for all $p, q \in \mathcal{C}_\Sigma$, if $p \rightsquigarrow \mu_p$ then there exists a μ_q such that $q \rightsquigarrow \mu_q$ and $\mu_p \equiv_{A,R} \mu_q$.

In [18], operational semantics for probabilistic systems is studied from a categorical viewpoint by Bartels and the results are translated in the form of concrete rule formats for probabilistic systems. In particular, Bartels defined the following rule format, called PGSOS (for Probabilistic GSOS) for reactive probabilistic systems.

Definition 56 ([Probabilistic GSOS Format) *A deduction rule is in the* PGSOS *format when it is of the following form.*

$$\frac{\{x_i \xrightarrow{a_i} y_i \mid i \in I\} \cup \{x_j \stackrel{a_j}{\not\rightarrow} \mid j \in J\} \cup \{x_k \xrightarrow{a_k[\rho_k]} y_k \mid 1 \leq i \leq ar(f), k \in K\}}{f(\overrightarrow{x}) \xrightarrow{a[w \cdot \prod_{k \in K} \rho_k]} t}$$

where (common to GSOS*) variables in* $\overrightarrow{x}$*,* y_i*'s and* y_k*'s are all pairwise distinct,* I *and* J *are subsets of* $\{i \mid 1 \leq i \leq ar(f)\}$*,* K *is a finite index set,* ρ_k*'s are pairwise distinct probability variables,* $w \in (0,1]$ *is the weight of the rule,* $vars(t) \subseteq vars(\overrightarrow{x}) \cup \{y_k \mid k \in K\}$ *and* $\{y_k \mid k \in K\} \subseteq vars(t)$*.*

A trigger of such a rule is a set E *such that* $\{a_i \mid i \in I\} \subseteq E$ *and* $E \cap \{a_j \mid j \in J\} = \emptyset$ *(there might be more than one trigger for each rule).*

A TSS in the PGSOS *format has a set of rules such that for all function symbols* f*, the number of* f*-rules with label* a *and a trigger* E *is finite and if there are any* f*-rules with label* a *and a trigger* E*, their probability weights sum up to* 1*.*

The following theorem specifies the properties that can be proven for TSS's conforming to the PGSOS format.

Theorem 57 *A TSS conforming to the* PGSOS *format, defines a reactive operational semantics (i.e., the probabilities of transitions with each label sum up to* 0 *or* 1*, also called* semi-stochasticity*), such that reactive probabilistic bisimilarity is a congruence and guarded recursive equations have a unique solution up to reactive probabilistic bisimilarity.*

In [78], the following rule format is defined for generative probabilistic systems.

Definition 58 (PB Format) *A rule is in the* PB *format when it has the following form.*

$$\frac{\{x_i \overset{a[\rho_i]}{\rightarrow} y_i \mid i \in I\} \cup \{x_j \overset{A_j[\rho_j]}{\not\rightarrow} \mid j \in J\} \cup \{x_k \overset{A_k}{\not\rightarrow} \mid k \in K\}}{f(\overrightarrow{x}) \overset{a[w.\frac{\prod_{i \in I} \rho_i}{\prod_{j \in J}(1-\rho_j)}]}{\longrightarrow} t}$$

where I*,* J *and* K *are subsets of* $\{i \mid 1 \leq i \leq ar(f)\}$*, (for all* $i \in I$*,* $j \in J$ *and* $k \in K$*) variables in* $\overrightarrow{x}$ *and* y_i*'s are all pairwise distinct,* $A_i \subseteq A$*,* $A_k \subseteq A$*, if* $i = j$ *then* $a_i \notin A_j$*,* ρ_i*'s and* ρ_j*'s are pairwise distinct probability variables (ranging over* $(0,1]$ *and* $[0,1)$*, respectively),* $w \in (0,1]$ *is the weight of the rule and* $vars(t) \subseteq vars(\overrightarrow{x}) \cup \{y_k \mid k \in K\}$*. The set* $\{x_i \overset{a[\rho_i]}{\rightarrow} y_i \mid i \in I\}$ *is called* the set of active premises *of the above rule.*

An ordered TSS in the PB *format is a TSS plus a stratification (an ordering) function* $\mathcal{R}$ *from* $\{0,\ldots,n\}$ *(for some* $n \in I\!N$*) to the set of rules such that for all* $i \leq n$*, for all* f*-rules in* $\mathcal{R}(i)$*:*

(1) the sets of premises $\{x_j \overset{A_j[\rho_j]}{\not\rightarrow} \mid j \in J\}$ *and* $\{x_k \overset{a_k}{\not\rightarrow} \mid k \in K\}$ *are the same;*
(2) the sets of variables $\{x_i \mid i \in I\}$ *are the same;*
(3) for all $j \in J$*, let* $U \subseteq I$ *be the set satisfying for all* $u \in U$*, if* $u = j$ *and*

$a_u \notin A_j$, then there exists an f-rule in $\mathcal{R}(i)$ with $\{x_u \overset{a_u[\rho_u]}{\rightarrow} y_u \mid u \in U\}$ as the set of its active premises.

(4) the weights of all f-rules in $\mathcal{R}(i)$ sum up to 1 (if any such rule exists).

The intuition behind premises of the form $x_j \overset{A_j[\rho_j]}{\rightarrow}$ is that the probabilities of transitions with labels in A_j sum up to ρ_j and $x_k \overset{A_k}{\rightarrow}$ means that at least one transition (with a non-zero probability) is possible with a label in A_k. The conditions on f-rules in the same stratum guarantee that f-rules in the same stratum are either all enabled or all disabled and using this, one can guarantee, using the last item, that the semantics is indeed semi-stochastic. The semantics induced by a TSS of the above form is defined in [78] and the following theorem summarizes the properties of such semantics.

Theorem 59 *For an ordered TSS in the* PB *format, the probabilities of outgoing transitions from each closed term sum up to 0 or 1 and the generative bisimilarity is a congruence.*

11 Expressiveness

As can be noted from Figure 1 and the discussions in Section 3, several syntactic ingredients can be present or absent in a particular syntactic rule format, and thus, it is interesting to know whether adding certain features will increase the expressive power of the format. In a more general sense, this question can be generalized to the question concerning the class of transition systems that a certain rule format can specify. Different approaches have been taken in answering this question.

The first expressiveness result was given by de Simone in [42] where he shows that any operator definable in his format can be defined in terms of SCCS-Meije operators up to strong bisimilarity. Also, in [109] Parrow shows that all operators definable by a restricted form of the de Simone format (called asynchronous rules) can be defined in terms of a few primitives (called modules) and two operators: a binary disjoint parallel composition $|$ and unary linking $\langle a, b \rangle$ which hides the result of synchronization.

In [24], it has been proved that the operational semantics induced by a TSS in the GSOS format is effective, i.e., for each state (of the countable many ones available) the number of outgoing transitions and the next states can be computed. In [137] a general theorem has been proven showing that there is no effective operational semantics on a enumerable syntax and with more than one label that can denote all effective process graphs up to trace equivalence (or any other stronger behavioral equivalence). It follows from this theorem that there exists an effective process graph that cannot be specified by any TSS in the GSOS format up to trace equivalence. There is a guardedness and finiteness assumption on the signature in the GSOS format which does not allow one to specify process graphs that are not guarded or effective. The effect of relaxing these assumptions is studied in [137].

Another major approach to expressiveness is to characterize the *finest completed-trace congruence* induced by operators definable using a certain rule format. The idea is to take an arbitrary language definable by a format with an operational semantics and measure the expressive power of the rule format by adding new operators to the language and checking how much we can tell the difference between old processes by putting them into newly defined contexts. In [24], it is shown that the finest completed-trace congruence induced by GSOS definable operators is ready simulation, thus the title of their paper "Bisimulation Can't be Traced". Groote and Vaandrager studied the same issue for the tyft/tyxt format and proved that the finest completed-trace congruence induced by definable tyft/tyxt operators is 2-nested simulation. For ntyft/ntyxt, it was shown in [64] that the finest completed-trace congruence is strong bisimilarity, which is a good indication of the added expressiveness of ntyft/ntyxt in comparison with its subsets GSOS and tyft/tyxt.

In [48], it is shown that the well-foundedness assumption can be relaxed from the ntyft/ntyxt format and for all non well-founded TSS's in the ntyft/ntyxt format there exists a TSS in the NTree format (defined below, with a potential infinite blow up in the number of rules) that defines the same three-valued model.

Definition 60 (NTree (Tree) Format) *A TSS in the ntyft/ntyxt (tyft/tyxt) format is in the NTree (Tree) format when the left-hand sides of the positive premises are variables and the variable dependency graphs of all rules are well-founded.*

Note that the left-hand side of negative premises can still contain composite terms and it has been shown in [48] that restricting the left-hand side of negative premises to variables, i.e., restricting to the so-called nxyft format, reduces the expressiveness in the sense that there exists a transition relation which is definable by a TSS in the ntyft/ntyxt (and thus the NTree) format but not by any TSS in the nxyft format.

In [103], it is shown that, unlike in the (n)tyft/(n)tyxt case, dropping the well-foundedness assumption from the promoted tyft format will jeopardize the congruence result. Furthermore, the expressiveness (i.e., the set of specifiable transition relations) of the tyft, the promoted tyft, and the positive subset of the promoted panth formats are compared in [103] and it is showed that while the tyft format with closed terms is incomparable to the promoted tyft format, the positive subset of the promoted panth format is strictly more expressive than both.

12 Reasoning Techniques

In [69], a logical framework, nowadays called *Hennessy-Milner logic* after the authors' names, is proposed. Hennessy-Milner logic can be used to reason about processes and characterize their equalities. The syntax of Hennessy-Milner logic (with infinite conjunction and without recursion) is defined as follows:

$$\Phi ::= \bigwedge_{i \in I} \Phi_i \mid \langle l \rangle \Phi \mid \neg \Phi$$

where I is an index set (possibly infinite) and $\perp$ stands for conjunction with $I = \emptyset$. Truth of a Hennessy-Milner formula $\phi \in \Phi$ is defined with respect to a particular process p, denoted by $p \vDash \phi$. The intuition behind $p \vDash \langle l \rangle \phi$ is that $\langle l \rangle \phi$ is true for p when there exists a p' such that $p \xrightarrow{l} p'$ and ϕ is true for p'. The intuitions behind conjunction and negation are as expected. In [89,70], it is shown that Hennessy-Milner logic is sound and adequate for strong bisimilarity, meaning that two processes are bisimilar if and only if they satisfy the same set of Hennessy-Milner formulae.

In [79,80] a meta-theory is developed that allows for decomposing Hennessy-Milner formulae using the structure of terms in a generic way by examining deduction rules of the process language in the de Simone format. This result has been improved in [49] and extended to both the ready simulation format (ntyft/ntyxt format without lookahead) and the tyft/tyxt format. The technique developed in [49] defines for each term t and each logical formula ϕ, a set of mappings $\psi : vars(t) \to \Phi$ from variables of term t to logical formulae such that ϕ is true for $p = \sigma(t)$ if and only if for at least one of such mappings, $\sigma(x) \vDash \psi(x)$ for all $x \in vars(t)$. Below, we quote a simple example from [49] to better illustrate the idea.

Example 61 *Consider the following deduction rules*

$$\frac{}{a.x \xrightarrow{a} x} \qquad \frac{x \xrightarrow{a} x'}{x \,||\, y \xrightarrow{a} x' \,||\, y} \qquad \frac{y \xrightarrow{a} y'}{x \,||\, y \xrightarrow{a} x \,||\, y'}$$

and a Hennessy-Milner formula $\phi = \langle a \rangle \langle b \rangle \top$. Then $p \,||\, q \vDash \phi$ if and only if at least one of the following cases holds:

- $p \vDash \langle a \rangle \langle b \rangle \top$;
- $q \vDash \langle a \rangle \langle b \rangle \top$;
- $p \vDash \langle a \rangle \top$ *and* $q \vDash \langle b \rangle \top$;

- $p \vDash \langle b \rangle \top$ *and* $q \vDash \langle a \rangle \top$.

For the decomposition to work effectively on variables, the rules in tyft/tyxt format and the rules in ready simulation format have to be reduced to rules in which the source of the premises are only variables. Thus, the notion of *nxytt*-ruloids is used (called P-ruloids in [49]; see Definition 30) to denote all derivable rules with variables as the source of premises (and in the case of the ready simulation format, without lookahead). The detailed construction for ruloids and their properties can be found in [49].

Common to the above example, the decomposition method defines for each process p, a function $-^{-1} : \mathcal{T}_\Sigma \to (\Phi \to \mathcal{P}(V \to \Phi))$ which for each term and logical formula, gives a set of mappings from variables to logical formulae:

- For $\phi = \langle l \rangle \phi'$, and an arbitrary term t, define $\psi : \mathit{vars}(t) \to \Phi \in t^{-1}(\phi)$ when there exists an *nxytt*-ruloid $\dfrac{H}{t \xrightarrow{l} u}$, and there exists a $\chi \in u^{-1}(\phi)$ as

$$\psi(x) = \begin{cases} \bigwedge_{x \xrightarrow{l'} y \in H} \langle l' \rangle \psi(y) \wedge \bigwedge_{x \overset{l''}{\nrightarrow} \in H} \neg \langle l'' \rangle \top \wedge \chi(x) & \text{if } x \in \mathit{vars}(u) \\ \bigwedge_{x \xrightarrow{l'} y \in H} \langle l' \rangle \psi(y) \wedge \bigwedge_{x \overset{l''}{\nrightarrow} \in H} \neg \langle l'' \rangle \top & \text{if } x \notin \mathit{vars}(u); \end{cases}$$

- For $\phi = \bigwedge_{i \in I} \phi_i$, and an arbitrary term t, define $\psi : \mathit{vars}(t) \to \Phi \in t^{-1}(\phi)$ when there are $\psi_i : \mathit{vars}(t) \to \Phi \in t^{-1}(\phi_i)$ (for all $i \in I$) as:

$$\psi(x) = \bigwedge_{i \in I} \psi_i(x);$$

- For $\phi = \neg \phi'$, and an arbitrary term t, define $\psi : \mathit{vars}(t) \to \Phi \in t^{-1}(\phi)$ when there exists a function $h : t^{-1}(\phi) \to \mathit{vars}(t)$ as:

$$\psi(x) = \bigwedge_{\chi \in h^{-1}(x)} \neg \chi(x).$$

In [50], the decomposition technique is used to derive congruence formats for the notion of eta-bisimilarity, as an example.

Simpson in [123] introduces a compositional proof system for checking Hennessy-Milner formulae on processes specified in the GSOS format. The proof calculus is in the natural deduction style and uses sequents that combine judgements about logical formulae (of the form $p : \phi$, meaning that p satisfies ϕ) and transition formulae (of the form $p \xrightarrow{l} q$). The deduction rules concerning Hennessy-

Milner formulae (restricted to final conjunction) are the following:

$$\frac{}{\Gamma \Longrightarrow p : \top} \qquad \frac{\Gamma, p : \phi \Longrightarrow \Delta}{\Gamma \Longrightarrow p : \neg\phi, \Delta} \qquad \frac{\Gamma, p : \neg\phi \Longrightarrow \Delta}{\Gamma \Longrightarrow p : \phi, \Delta}$$

$$\frac{\Gamma, p : \phi, p : \psi \Longrightarrow \Delta}{\Gamma, p : \phi \wedge \psi \Longrightarrow \Delta} \qquad \frac{\Gamma \Longrightarrow p : \phi, p : \psi, \Delta}{\Gamma \Longrightarrow p : \phi \wedge \psi, \Delta}$$

$$\frac{\Gamma, p \xrightarrow{l} x, x : \phi \Longrightarrow \Delta}{\Gamma, p : \langle l \rangle \phi \Longrightarrow \Delta} \qquad \frac{\Gamma \Longrightarrow p \xrightarrow{l} q, \Delta \quad \Gamma \Longrightarrow, q : \phi, \Delta}{\Gamma \Longrightarrow p : \langle l \rangle \phi, \Delta}$$

where in the rule concerning the introduction of $\langle l \rangle \phi$ on the left-hand-side, x should not appear anywhere else in the rule.

The following rules concerning transition formulae are mainly derived from the corresponding GSOS rules and the intuition behind transition formulae.

$$\frac{\Gamma, p \xrightarrow{l} q \Longrightarrow \Delta}{\Gamma \Longrightarrow p \overset{l}{\nrightarrow} q, \Delta} \qquad \frac{\Gamma, p \overset{l}{\nrightarrow} \Longrightarrow \Delta}{\Gamma \Longrightarrow p \xrightarrow{l} q, \Delta} \qquad \frac{}{\Gamma, x \xrightarrow{l} y \Longrightarrow x \overset{l}{\nrightarrow} y, \Delta}$$

$$\frac{\{\Gamma \Longrightarrow p_i \xrightarrow{l_{ij}} q_{ij}, \Delta \mid i \in I, 1 \leq j \leq m_i\}, \{\Gamma \Longrightarrow p_j \overset{l_{jk}}{\nrightarrow}, \Delta \mid j \in J, 1 \leq k \leq n_j\}}{\Gamma \Longrightarrow f(\overrightarrow{p}) \xrightarrow{l} t[\vec{x} \mapsto \vec{p}, y_{ij} \mapsto q_{ij}], \Delta}$$

$$\frac{\begin{array}{r}\Gamma[t[\vec{x} \mapsto \vec{p}]/x], \{p_i \xrightarrow{l_{ij}} q_{ij} \mid i \in I, 1 \leq j \leq m_i\}, \{p_j \overset{l_{jk}}{\nrightarrow} \mid j \in J, 1 \leq k \leq n_j\} \\ \Longrightarrow \Delta[t[\vec{x} \mapsto \vec{p}]/x]\end{array}}{\Gamma, f(\overrightarrow{p}) \xrightarrow{l} x \Longrightarrow \Delta}$$

where the above rules concerning the transition of $f(\vec{p})$ are present for all (lists of) closed terms $\vec{p}$, and q_{ij} and all deduction rules of the following form in the TSS.

$$\frac{\{x_i \xrightarrow{l_{ij}} y_{ij} \mid i \in I, 1 \leq j \leq m_i\} \cup \{x_j \overset{l_{jk}}{\nrightarrow} \mid j \in J, 1 \leq k \leq n_j\}}{f(\overrightarrow{x}) \xrightarrow{l} t}$$

Soundness and completeness (relative to a class of assumptions) are proved in [123] for the above proof system.

13 Implementation

13.1 Term Rewriting

Several authors have studied the, rather evident, link between rewriting logic [83] and SOS both from a theoretical [43,83–85] as well as practical point of view [30,31,140,142].

In [30], the outline of a translation from Modular SOS (MSOS) [97,96] to the Maude rewriting logic is given and proven correct. The translation is quite straightforward and the main technical twist is in the decomposition of labels into the configurations in the source and the targets of the transitions which is due to the structure of labels in MSOS. The translation is fully implemented and details of this implementation can be consulted in [29]. [32] reports an extension of the Maude Modular SOS Tool (MMT) with Maude's Strategy Language (MSL) which allows for defining strategies for animating (M)SOS specifications. An example of such strategies is given for the case of Ordered SOS specifications.

Verdejo in [141] and Verdejo and Marti-Oliet in [139,140,142] report the implementation of a number of instances of SOS semantics in Maude [36]. Our approach is very close in essence to their work in that SOS deduction rules are interpreted as Maude conditional rewrite rules. In [102], some aspects of SOS meta-theory are implemented in Maude. The implementation defines a GSOS-based framework and provides a way to check the premises of congruence and operational conservativity meta-theorems. Furthermore, it allows for animating programs based on their GSOS specifications.

13.2 Logic Programming (Centaur)

Centaur [26] is a tool developed in the 1980's within the group of the late Gilles Kahn at INRIA Sophia-Antipolis. It is described as a "generic interactive environment generator", i.e., it provides an environment for generating and integrating syntactic manipulation and semantic animation tools. Centaur has been widely used by different language developers (at over one hundred sites). The high-level architecture of Centaur is depicted in Figure 2.

The abstract syntax of languages can be specified in Metal [74] or ASF [33]. Then the Centaur tool generates a parser (an incremental parser in the case of ASF specifications) which is an independent process and can interact further with the tool-set. Pretty-printing instructions can be specified in the PPML formalism [95] and the result of compiling the PPML specification can be

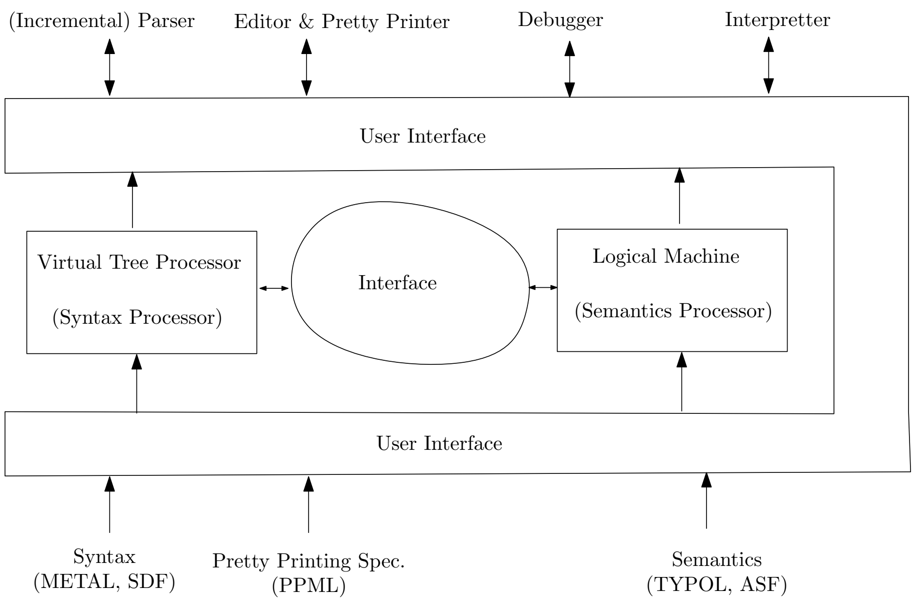

Fig. 2. High-Level Architecture of Centaur

used to pretty print the syntax in the editor. Semantics of languages can be specified in Typol which is a textual description of the natural semantics [73] which can be used to generate interpreters and debuggers.

The syntactic manipulation part is implemented in Le-Lisp and the logical machine is written in Prolog. The interface between the two modules is written in the C programming language.

13.3 Functional Programming (LETOS, PAC)

LETOS [67] is a tool that generates LaTeX documents as well as executable animation code in Miranda [129] from a wide range of semantics, including some forms of SOS. A first attempt to implement an SOS meta-theorem, concerning operational conservativity of [65] is also reported in [67]. However, the implementation does not fully check this theorem and only checks the *source-dependency* requirement which is one of the hypotheses of the conservativity theorem of [65].

In [35] an approach for implementing SOS rules is presented which combines (unconditional) term-rewriting and λ-calculus. By putting some extra conditions on the syntax of the SOS (which are satisfied by GSOS rules), they prove that the implemented rewrite system is one-step sound and complete as well as divergence sound and complete with respect to the transition system induced by the SOS. Furthermore, they prove confluence and termination of their rewrite systems (for the micro-steps on the way to making a complete operational transition). In [34] an implementation of the approach in the Larch

Prover [66] is reported.

Process Algebra Compiler (PAC) [37] is a tool that takes the signature and the SOS rules of a language and generates a scanner/parser as well as verification libraries targeted at the specified language. The generated libraries can then be compiled in combination with the kernels of a verification tool in order to generate a complete verifier for the language with the specified syntax and semantics. The scanner/parser specification are generated in LEX/YACC syntax and the verification libraries can be generated both in Lisp and in Standard ML which can then be compiled with the kernels of the MAUTO tool [28] and the Concurrency Workbench [38], respectively.

14 Other Meta-Results

14.1 Non-Interference

Confidentiality is an important aspect of security and non-interference [63] is a well-studied means to guarantee end-to-end confidentiality. Non-interference means that a user with a lower confidentiality level cannot infer anything about the higher-level information by interacting with the system (using lower-level methods that it has in hand). In [126,127] a rule format for non-interference is proposed which is based on the Cool languages format (in order to guarantee compositionality of non-interference) and imposes further restrictions to assure that the lower-level behavior of the system does not change as a result of performing higher-level transitions.

14.2 Reversibility

Reversible computations are known to have some interesting properties (such as being potentially dissipation free) and they can model several real-world phenomena such as biological systems [40], chemical reactions [21] and quantum computation [118]. In [112], a general recipe is given for reversing the semantics in the SOS style. The technique is based on a subset of the de Simone format and generates a new SOS which has a reverse transition relation $\rightsquigarrow$ and new predicates to recognize reversible terms from terms in the original syntax and to generate fresh tags (in order to distinguish different instances of the same action in concurrency scenarios). We illustrate the approach with a simple example below.

Example 62 *Consider the following standard deduction rules*

$$\frac{}{a.x \xrightarrow{a} x} \qquad \frac{x \xrightarrow{a} x'}{x \,||\, y \xrightarrow{a} x' \,||\, y} \qquad \frac{y \xrightarrow{a} y'}{x \,||\, y \xrightarrow{a} x \,||\, y'} \qquad \frac{x \xrightarrow{a} x' \quad y \xrightarrow{\overline{a}} y'}{x \,||\, y \xrightarrow{\tau} x' \,||\, y'}$$

defined on a signature with a constant 0*, action prefixing* $a.\_$ *and nondeterministic choice* $\_+\_$*. The reversible model of the above semantics and in particular the reverse transition relation* $\rightsquigarrow$ *is defined as follows.*

$$\frac{std(x)}{a.x \xrightarrow{a[n]} a[n].x} \qquad \frac{std(x)}{a[n].x \overset{a[n]}{\rightsquigarrow} a.x} \qquad \frac{x \xrightarrow{b[m]} x'}{a[n].x \xrightarrow{b[m]} a[n].x'} \qquad \frac{x \overset{b[m]}{\rightsquigarrow} x'}{a[n].x \overset{b[m]}{\rightsquigarrow} a[n].x'}$$

$$\frac{x \xrightarrow{a[n]} x' \quad fresh_n(y)}{x \,||\, y \xrightarrow{a[n]} x' \,||\, y} \qquad \frac{x \overset{a[n]}{\rightsquigarrow} x' \quad fresh_n(y)}{x \,||\, y \overset{a[n]}{\rightsquigarrow} x' \,||\, y} \qquad \frac{y \xrightarrow{a[n]} y' \quad fresh_n(x)}{x \,||\, y \xrightarrow{a[n]} x \,||\, y'}$$

$$\frac{y \overset{a[n]}{\rightsquigarrow} y' \quad fresh_n(x)}{x \,||\, y \overset{a[n]}{\rightsquigarrow} x \,||\, y'} \qquad \frac{x \xrightarrow{a[n]} x' \quad y \xrightarrow{\overline{a}[n]} y'}{x \,||\, y \xrightarrow{\tau[n]} x' \,||\, y'} \qquad \frac{x \overset{a[n]}{\rightsquigarrow} x' \quad y \overset{\overline{a}[n]}{\rightsquigarrow} y'}{x \,||\, y \overset{\tau[n]}{\rightsquigarrow} x' \,||\, y'}$$

where the predicates $std(p)$ *and* $fresh_n(p)$ *check whether term* p *is standard (belongs to the original syntax, i.e., does not contain any tagged action) and* $fresh_n(p)$ *checks whether the tag* n *does not appear (is fresh) for* p*. Both predicates can be defined structurally on terms but we omit their semantics here.*

In [112], the following results are proved for the transformed reversible semantics. In the remainder of this section, μ and ν stand for labels of the form $a[n]$, $\overline{b}[m]$ or $\tau[k]$ and $\xrightarrow{\mu^*}$ stands for a number (zero or more) transition with the above-mentioned labels.

(1) Reversibility: $p \xrightarrow{\mu} q$ if and only if $q \overset{\mu}{\rightsquigarrow} p$;
(2) Well-foundedness: The reversible transition relation is well-founded (no infinite chain of backward transitions exists) since backward computation can only reverse the forward steps made so far (which are finite);
(3) Unique Transitions: For all forward and backward computations $p \overset{\mu}{\mapsto} q$ and $p \overset{\nu}{\mapsto} q$ (where $\mapsto$ can be $\rightarrow$ or $\rightsquigarrow$), it holds that $\mu = \nu$. This property holds because labels are annotated with unique tags and after the transition the tags are recorded in the syntax;
(4) Reverse diamond: if $p \overset{a[m]}{\rightsquigarrow} q$ and $p \overset{b[n]}{\rightsquigarrow} r$ and $m \neq n$, there exists an s such that $q \overset{b[n]}{\rightsquigarrow} s$ and $r \overset{a[m]}{\rightsquigarrow} s$;
(5) Forward diamond: if $p \xrightarrow{a[m]} q \xrightarrow{\mu^*} t$ and $p \xrightarrow{b[n]} r \xrightarrow{\nu^*} t$ and $m \neq n$, there exists an s such that $q \xrightarrow{b[n]} s$ and $r \xrightarrow{a[m]} s$ and $s \xrightarrow{\mu^* \setminus b[n]} t$ and $s \xrightarrow{\nu^* \setminus a[m]} t$ (where $\mu^* \setminus a[n]$ denotes removing the instances of $a[n]$ in μ^*);

(6) Forward-reverse bisimilarity in the transformed reversible model is a congruence and is included in the strong bisimilarity in the original model. Forward-reverse bisimilarity is defined below.

Definition 63 (Forward-Reverse Bisimilarity) *A symmetric relation $R \subseteq \mathcal{C}_\Sigma \times \mathcal{C}_\Sigma$ is a bisimulation relation w.r.t. a forward transition relation $\rightarrow$, a backward transition relation $\rightsquigarrow$, and a predicate $\downarrow$ when for all $p, q \in \mathcal{C}_\Sigma$ such that $(p, q) \in R$, it satisfies*

- *for all $p' \in \mathcal{C}_\Sigma$ and $l \in L$, if $p \xrightarrow{l} p'$, then $q \xrightarrow{l} q'$ for some $q' \in \mathcal{C}_\Sigma$ such that $(p', q') \in R$;*
- *for all $p' \in \mathcal{C}_\Sigma$ and $l \in L$, if $p \overset{l}{\rightsquigarrow} p'$, then $q \overset{l}{\rightsquigarrow} q'$ for some $q' \in \mathcal{C}_\Sigma$ such that $(p', q') \in R$;*
- *$(l) \downarrow\ p$ if and only if $(l) \downarrow\ q$.*

Two closed terms p and q are forward-reverse bisimilar if and only if there exists a bisimulation relation R such that $(p, q) \in R$.

14.3 Bounded nondeterminism

In [137], a rule format is proposed, by imposing restrictions on the de Simone format, which guarantees that the induced semantics affords only bounded non-determinism, i.e., each closed term has only finite number of outgoing transitions. Fokkink and Vu in [52] generalize the result of [137] to a far more general SOS framework.

Bloom (reference [5] from the paper [52]) generalized GSOS to a higher-order setting and formulated a notion of boundedness reminiscent to the one of [137]: Bounded TSS in higher-order GSOS is bounded nondeterministic.

The following definitions are from [52]. Note that only TSS's with positive premises are considered and there is only one transition relation. The semantics is three-valued stable models.

Definition 64 (Bounded Nondeterminism) *An LTS is bounded nondeterministic if all its states are finitely branching. A state s in an LTS is finitely branching if it has only finitely many outgoing transitions.*

Definition 65 (Bounded Nondeterminism Format) *A transition rule is in bounded nondeterminism format if: (1) all variables occurring in the left-hand side of its positive premises also occur in its source, and (2) all variables occurring in its target also occur in its source or in the right-hand side of a positive premise.*

A TSS is in bounded nondeterminism format if all its transition rules are.

Definition 66 (η-Type) *Let R be a TSS. Let η map each term in $\mathcal{T}_\Sigma(\Sigma)$ to a finite subset of $\mathcal{T}_\Sigma(\Sigma)$. Consider a transition rule ρ with source t and positive premises $\{t_i \xrightarrow{l_i} t_i' \mid i \in I\}$. The mapping $\varphi : \eta(t) \to \mathcal{P}_\omega(\mathcal{T}_\Sigma(\Sigma))$ is defined as follows:*

$$\varphi(u) = \{l_i \mid i \in I \wedge t_i = u\}.$$

$\langle t, \varphi \rangle$ is said to be the η-type of the transition rule ρ.

Definition 67 (Uniform) *A TSS is uniform if sources of transition rules that are α-convertible are always syntactically equal.*

Definition 68 (Bounded) *A TSS is bounded if it is uniform and for each η-type the corresponding set of transition rules is finite. A TSS is uniform if sources of transition rules that are α-convertible are always syntactically equal.*

Theorem 69 *If a TSS is bounded, in bounded nondeterminism format and strictly stratifiable, then the LTS associated to the TSS is bounded nondeterministic.*

References

[1] S. Abramsky. The Lazy λ-Calculus. In D.A. Turner, editor, *Research Topics in Functional Programming*, pages 65-116. Addison-Wesley, 1990.

[2] L. Aceto. Deriving complete inference systems for a class of GSOS languages generating regular behaviours. In B. Jonsson and J. Parrow, editors, *Proceedings of the 5th International Conference on Concurrency Theory (CONCUR'94)*, volume 836 of LNCS, pages 449-464. Springer, 1994.

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

[4] Luca Aceto and Anna Ingólfsdóttir. CPO models for GSOS languages — Part I: Compact GSOS languages. *Information and Computation*, 129(2):107–141, 1994.

[5] L. Aceto, W.J. Fokkink, and C. Verhoef. Structural operational semantics. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra, Chapter 3*, pages 197-292. Elsevier, 2001.

[6] E.A. Astesiano, A. Giovini, and G. Reggio. Generalized bisimulation in relational specifications. In R. Cori and M. Wirsing, editors, *Proceedings of the 5th Annual Symposium on Theoretical Aspects of Computer Science (STACS'88)*, volume 294 of LNCS, pages 207-226. Springer, 1988.

[7] J.C.M. Baeten and J.A. Bergstra. Processen en procesexpressies. *Informatie*, 30(3):214-222, 1988.

[8] J.C.M. Baeten and J.A. Bergstra. Real time process algebra. *Formal Aspects of Computing*, 3:142-188, 1991.

[9] J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. Syntax and defining equations for an interrupt mechanism in process algebra. *Fundamenta Informaticae*, IX:127-168, 1986.

[10] J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. Verification of an alternating bit protocol by means of process algebra, In W. Bibel and K.P. Jantke, editors, *Proceedings Mathematical Models of Specification and Synthesis of Software Systems 1985*, volume 215 of LNCS, pages 9-23. Springer, 1986.

[11] J.C.M. Baeten, J.A. Bergstra, and J.L.M. Vrancken. Processen en procesexpressies. Technical report P8705, Universiteit van Amsterdam, 1987.

[12] J.C.M. Baeten and R.J. van Glabbeek. Merge and termination in process algebra. In K.V. Nori, editor, *Proceeding of the 7th Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS'05)*, volume 287 of LNCS, pages 153-172. Springer, 1987.

[13] J.C.M. Baeten and C.A. Middelburg. *Process Algebra with Timing.* EATCS Monographs. Springer, 2002.

[14] J.C.M. Baeten and C. Verhoef. A congruence theorem for structured operational semantics with predicates. In Eike Best, editor, *Proceedings of the 4th International Conference on Concurrency Theory (CONCUR'93)*, volume 715 of LNCS, pages 477-492. Springer, 1993.

[15] J.C.M. Baeten and E.P. de Vink. Axiomatizing GSOS with termination. *Journal of Logic and Algebraic Programming*, 60-61:323-351, 2004.

[16] J.C.M. Baeten and W.P. Weijland. *Process Algebra.* volume 18 of Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, 1990.

[17] F. Bartels, A. Sokolova and E.P. de Vink. A hierarchy of probabilistic system types. *Theoretical Computer Science*, 327(1-2):3-22, 2004.

[18] F. Bartels. GSOS for probabilistic transition systems. In *Proceedings of the 5th International Workshop on Coalgebraic Methods in Computer Science (CMCS'02)*, volume 65 of ENTCS, pages 1-25. Elsevier, 2002.

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

[20] K.L. Bernstein. A congruence theorem for structured operational semantics of higher-order languages. In *Proceedings of the 13th IEEE Symposium on Logic In Computer Science (LICS'98)*, pages 153-164. IEEE CS, 1998.

[21] G. Berry and G. Boudol. The Chemical Abstract Machine. *Theoretical Computer Science*, 96(1):217-248, 1992.

[22] B. Bloom. Structural operational semantics for weak bisimulations. *Theoretical Computer Science*, 146:25-68, 1995.

[23] B. Bloom, W.J. Fokkink, and R.J. van Glabbeek. Precongruence formats for decorated trace semantics. *ACM Transactions on Computational Logic*, 5(1):26-78, 2004.

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

[25] R.N. Bol and J.F. Groote. The meaning of negative premises in transition system specifications. *Journal of the ACM*, 43(5):863-914, 1996.

[26] P. Borras, D. Clément, T. Despeyroux, J. Incerpi, G. Kahn, B. Lang, and V. Pascual. CENTAUR: The system. *SIGPLAN Notices*, 24(2):14-24, 1988.

[27] G. Boudol. Towards a lambda-calculus for concurrent and communicating systems. In J. Díaz and F. Orejas, editors, *Proceedings of the International Joint Conference on Theory and Practice of Software Development (TAPSOFT'89)*, volume 351 of LNCS, pages 149-161. Springer, 1989.

[28] G. Boudol, V. Roy, R. de Simone, and D. Vergamini. Process calculi from theory to practice: Verification tools. In *Proceedings of the Workshop on Automated Verification Methods for Finite State Systems*, volume 407 of *LNCS*, pages 1-10. Springer, 1989.

[29] C. de O. Braga. *Rewriting Logic as a Semantic Framework for Modular Structural Operational Semantics.* PhD thesis, Departamento de Informática, Pontifícia Universidade Católica de Rio de Janeiro, Brasil, 2001. `http://www.ic.uff.br/~cbraga`.

[30] C. de O. Braga, E.H. Haeusler, J. Meseguer, and P.D. Mosses. Mapping modular SOS to rewriting logic. In M. Leuschel, editor, *Proceedings of the 12th International Workshop on Logic Based Program Synthesis and Transformation (LOPSTR'02)*, volume 2664 of LNCS, pages 262-277. Springer, 2002.

[31] C. de O. Braga, E.H. Haeusler, J. Meseguer, and P.D. Mosses. Maude action tool: Using reflection to map action semantics to rewriting logic. In T. Rus, editor, *Proceedings of the 8th International Conference on Algebraic Methodology and Software Technology (AMAST'00)*, volume 1816 of LNCS, pages 407-421. Springer, 2000.

[32] C. de O. Braga and A. Verdejo Modular SOS with strategies. In R.J. van Glabbeek and P.D. Mosses, editors, *Proceedings of the 3rd Workshop on Structural Operational Semantics (SOS'06)*, Electronic Notes in Computer Science, Elsevier, 2006. To appear.

[33] M. van den Brand, J. Heering, P. Klint and P.A. Olivier. Compiling language definitions: the ASF+SDF compiler. *ACM Transactions on Programming Languages and Systems* 24(4):334-368, 2002.

[34] K.-H. Buth. Using SOS definitions in term rewriting proofs. In U. Martin and J.M. Wing, editors, *Proceedings of the 1st International Workshop on Larch*, Workshops in Computing, pages 36-54. Springer, 1994.

[35] K.-H. Buth. Simulation of SOS definitions with term rewriting systems. In D. Sannella, editor, *Proceedings of the 5th European Symposium on Programming Languages and Systems (ESOP'94)*, volume 788 of LNCS, pages 150-164. Springer, 1994.

[36] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. The Maude 2.0 system. In R. Nieuwenhuis, editors, *Proceeding of the 14th International Conference on Rewriting Techniques and Applications (RTA'03)*, volume 2706 of LNCS, pages 76-87. Springer, 2003.

[37] R. Cleaveland, E. Madelaine, and S.Sims. Front-end generator for verification tools. In *Proceedings of the 1st International Workshop on Tools and Algorithms for Construction and Analysis of Systems (TACAS'95)*, volume 1019 of LNCS, pages 153-173, Springer, 1995.

[38] R. Cleaveland, J. Parrow, and B. Steffen. The Concurrency Workbench: A semantics-based tool for the verification of concurrent systems. *ACM Transactions on Programming Languages and Systems*, 15(1): 36-72, 1993.

[39] A. Corradini, R. Heckel, and U. Montanari. Compositional SOS and beyond: A coalgebraic view of open systems. *Theoretical Computer Science*, 280(1-2):163-192, 2002.

[40] V. Danos and C. Laneve. Formal molecular biology. *Theoretical Computer Science*, 325(1):69-110, 2004.

[41] R. de Simone. *Calculabilité et expressivité dans l'algèbre de processus parallèles Meije*, Thèse de 3^{e} cycle, Univ. Paris 7, 1984.

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

[43] P. Degano, F. Gadducci, and C. Priami. A causal semantics for CCS via rewriting logic. *Theoretical Computer Science*, 275(1-2):259-282, 2002.

[44] P. Degano and C. Priami. Enhanced operational semantics: a tool for describing and analyzing concurrent systems. *ACM Computing Surveys*, 33(2):135-176, 2001.

[45] M.P. Fiore and S. Staton. A Congruence rule format for name-passing process calculi from mathematical operational semantics. In *Proceedings of the 21st Symposium on Logic in Computer Science (LICS'06)*, pages 49-58. IEEE CS, 2006.

[46] W.J. Fokkink. The tyft/tyxt format reduces to tree rules. In M. Hagiya and J.C. Mitchell, editors, *Proceedings of the Symposium on Theoretical Aspects of Computer Software (STACS'94)*, volume 789 of LNCS, pages 440-453. Springer, 1994.

[47] W.J. Fokkink. Rooted branching bisimulation as a congruence. *Journal of Computer and System Science*, 60(1):13-37, 2000.

[48] W.J. Fokkink and R.J. van Glabbeek. Ntyft/ntyxt rules reduce to ntree rules. *Information and Computation*, 126(1):1-10, 1996.

[49] W.J. Fokkink, R.J. van Glabbeek, and P. de Wind. Compositionality of Hennessy-Milner logic by structural operational semantics. *Theoretical Computer Science*, 354(3):421-440, 2006.

[50] W.J. Fokkink, R.J. van Glabbeek and P. de Wind. Divide and congruence applied to eta-bisimulation. In P.D. Mosses and I. Ulidowski, editors, *Proceedings of the 2nd Workshop on Structural Operational Semantics (SOS 2005)*, volume 156(1) of ENTCS, pages 97-113. Elsevier, 2005.

[51] W.J. Fokkink and C. Verhoef. A conservative look at operational semantics with variable binding. *Information and Computation*, 146(1):24-54, 1998.

[52] W.J. Fokkink and T.D. Vu. Structural operational semantics and bounded nondeterminism. *Acta Informatica*, 39(6-7):501-516, 2003.

[53] M.J. Gabbay and A.M. Pitts. A new approach to abstract syntax involving binders. In *Proceedings of the 14th Annual Symposium on Logic in Computer Science (LICS'99)*, pages 214-224. IEEE CS, 1999.

[54] V. Galpin. A format for semantic equivalence comparison. *Theoretical Computer Science*, 309(1-3):65-109, 2003.

[55] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R.A. Kowalski and K.A. Bowen, editors, *Proceedings of the 5th International Conference on Logic Programming (ICLP'88)*, pages 1070-1080. MIT Press, 1988.

[56] R.J. van Glabbeek. Bounded nondeterminism and the approximation induction principle in process algebra. In F.-J. Brandenburg, G. Vidal-Naquet, and M. Wirsing, editors, *Proceedings of the 4th Annual Symposium on Theoretical Aspects of Computer Science (STACS'87)*, volume 247 of LNCS, pages 336-347. Springer, 1987.

[57] R.J. van Glabbeek. The linear time - branching time spectrum II. In Eike Best, editor, *Proceedings of the 4th International Conference on Concurrency Theory (CONCUR'93)*, volume 715 of LNCS, pages 66-81. Springer, 1993.

[58] R.J. van Glabbeek. Full abstraction in structural operational semantics (extended abstract). In M. Nivat, C. Rattray, T. Rus, and G. Scollo, editors, *Proceedings of the 3rd International Conference on Algebraic Methodology and Software Technology (AMAST '93)*, pages 75-82. Springer, 1993.

[59] R.J. van Glabbeek. The linear time - branching time spectrum I. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra, Chapter 1*, pages 3-100. Elsevier, 2001.

[60] R.J. van Glabbeek. The meaning of negative premises in transition system specifications II. *Journal of Logic and Algebraic Programming*, 60-61:229-258, 2004.

[61] R.J. van Glabbeek. On cool congruence formats for weak bisimulations (extended abstract). In D.V. Hung and M. Wirsing, editors, *Proceedings of the 2nd International Colloquium on Theoretical Aspects of Computing (ICTAC'05)*, volume 3722 of LNCS, pages 331-346. Springer, 2005.

[62] R.J. van Glabbeek, S.A. Smolka, and B. Steffen. Reactive, generative and stratified models of probabilistic processes. Information and Computation, 121(1):59-80, 1995.

[63] J.A. Goguen and J. Meseguer. Security policies and security models. In *IEEE Symposium on Security and Privacy*, pages 11-20. IEEE CS, 1982.

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

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

[66] J.V. Guttag and J.J. Horning. *Larch: languages and tools for formal specification.* Springer, 1993.

[67] P.H. Hartel. LETOS - a lightweight execution tool for operational semantics. *Software, Practice and Experience*, 29(15):1379-1416, 1999.

[68] M.C.B. Hennessy and G.D. Plotkin. Full abstraction for a simple parallel programming language. In J. Becvár, editor, *Proceedings of the 8th Symposium on Mathematical Foundations of Computer Science (MFCS79)*, volume 74 of LNCS, pages 108-120. Springer, 1979.

[69] M.C.B. Hennessy and R. Milner. Algebraic laws for non-determinism and concurrency. *Journal of the ACM*, 32(1):137-161, 1985.

[70] M.C.B. Hennessy and C. Stirling. The power of the future perfect in program logics. *Information and Control*, 67(1-3):23-52, 1985.

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

[72] D.J. Howe. Proving congruence of bisimulation in functional programming languages. *Information and Computation*, 124(2):103-112, 1996.

[73] G. Kahn. Natural semantics. In *Proceedings of 4th Annual Symposium on Theoretical Aspects of Computer Science (STACS'87)*, volume 247 of LNCS, pages 22-39. Springer, 1987.

[74] G. Kahn, B. Lang, B. Melese and E. Morcos. Metal: A formalism to specify formalisms. *Science of Computer Programming* 3(2):151-188, 1983.

[75] P.C. Kanellakis and S.A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86(1):43-68, 1990.

[76] M. Kick. *Coalgebraic modelling of timed processes.* Ph.D. Thesis, LFCS, School of Informatics, University of Edinburgh, 2002.

[77] B. Klin. From bialgebraic semantics to congruence formats. In L. Aceto, W.J. Fokkink and I. Ulidowski, editors, *Proceedings of the Workshop on Structural Operational Semantics (SOS'04)*, volume 128 of ENTCS, pages 3-37, Elsevier, 2005.

[78] R. Lanotte and S. Tini. Probabilistic congruence for semistochastic generative processes. In Vladimiro Sassone, editor, *Proceedings of the 8th International Conference on Foundations of Software Science and Computational Structures (FOSSACS'05)*, volume 3441 of LNCS, pages 63-78. Springer, 2005.

[79] K.G. Larsen. *Context-dependent bisimulation between processes.* Ph.D. Thesis, University of Edinburgh, 1986.

[80] K.G. Larsen and L. Xinxin. Compositionality through an operational semantics of contexts. *Journal of Logic and Computation*, 1(6):761-795, 1991.

[81] K.G. Larsen and A. Skou. Bisimulation through Probabilistic Testing. *Information and Computation*, 94(1):1-28, 1991.

[82] S.P. Luttik. *Choice quantification in process algebra.* Ph.D. Thesis, Department of Computer Science, University of Amsterdam, 2002.

[83] N. Martí-Oliet and J. Meseguer. Rewriting logic as a logical and semantic framework. In D.M. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, volume 9, pages 1-87. Kluwer, 2002.

[84] J. Meseguer. Conditioned rewriting logic as a united model of concurrency. *Theoretical Computer Science (TCS)*, 96(1):73-155, 1992.

[85] J. Meseguer and C. de O. Braga. Modular rewriting semantics of programming languages. In C. Rattray, S. Maharaj, and C. Shankland, editors, *Proceedings of the 10th International Conference on Algebraic Methodology and Software Technology (AMAST'04)*, volume 3116 of LNCS, pages 364-378. Springer, 2004.

[86] C.A. Middelburg. Variable binding operators in transition system specifications. *Journal of Logic and Algebraic Programming*, 47(1):15-45, 2001.

[87] C.A. Middelburg. An alternative formulation of operational conservativity with binding terms. *Journal of Logic and Algebraic Programming*, 55(1-2):1-19, 2003.

[88] D. Miller and A. Tiu. A proof theory for generic judgments. *ACM Transactions on Computational Logic* 6(4):749-783, 2005.

[89] R. Milner. A Modal characterisation of observable machine-behaviour. In E. Astesiano and C. Böhm, editors, *Proceedings of the 6th Colloquium on Trees in Algebra and Programming (CAAP'81)*, volume 112 of LNCS, pages 25-34. Springer, 1981.

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

[91] R. Milner. The polyadic π-calculus: a tutorial. In F.L. Bauer, W. Brauer, and H. Schwichtenberg, editors, *Logic and Algebra of Specification*, pages 203-246. Springer, 1993.

[92] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, part I. *Information and Computation*, 100(1):1-40, 1992.

[93] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, part II. *Information and Computation*, 100(1):41-77, 1992.

[94] F. Moller and C.M.N. Tofts. A temporal calculus of communicating systems. In J.C.M. Baeten and J.W. Klop, editors, *Proceedings of the Conference on Theories of Concurrency: Unification and Extension (CONCUR '90)*, volume 458 of LNCS, pages 401-415, Springer, 1990.

[95] E. Morcos-Chounet and A. Conchon. PPML: A general formalism to specify prettyprinting. In H.-J. Kugler, editor, *Proceedings of IFIP Congress*, pages 583-590. North-Holland, 1986.

[96] P.D. Mosses, Exploiting labels in structural operational semantics. *Fundamenta Informaticae*, 60(1-4):17-31, 2004.

[97] P.D. Mosses, Modular structural operational semantics. *Journal of Logic and Algebraic Programming*, 60-61:195-228, 2004.

[98] M.R. Mousavi, M.J. Gabbay, and M.A. Reniers. SOS for higher order processes. In *Proceedings of the 16th International Conference on Concurrency Theory (CONCUR'05)*, volume 3653 of LNCS, pages 308-322. Springer, 2005.

[99] M.R. Mousavi, M.R. Mousavi, I.C.C. Phillips, M.A. Reniers, I. Ulidowski. The meaning of ordered SOS. In *Proceedings of the 26th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'06)*, volume 4337 of LNCS, pages 334–345, Springer, 2006.

[100] M.R. Mousavi and M.A. Reniers. Congruence for structural congruences. In *Proceedings of the 8th International Conference on Foundations of Software Science and Computation Structures (FOSSACS'05)*, volume 3441 of LNCS, pages 47-62. Springer, 2005.

[101] M.R. Mousavi and M.A. Reniers. Orthogonal extensions in structural operational semantics. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP'05)*, volume 3580 of LNCS, pages 1214-1225. Springer, 2005.

[102] M.R. Mousavi and M.A. Reniers. Prototyping SOS meta-theory in Maude. In P.D. Mosses and I. Ulidowski, editors, *Proceedings of the 2nd Workshop on Structural Operational Semantics (SOS'05)*, volume 156(1) of ENTCS, pages 135-150. Elsevier, 2005.

[103] M.R. Mousavi and M.A. Reniers. On well-foundedness and expressiveness of promoted tyft. In R.J. van Glabbeek and P.D. Mosses, editors, *Proceedings of the 3rd Workshop on Structural Operational Semantics (SOS'06)*, Electronic Notes in Computer Science, Elsevier, 2006. To appear.

[104] M.R. Mousavi, M.A. Reniers, and J.F. Groote. Notions of bisimulation and congruence formats for SOS with data. *Information and Computation*, 200(1):104-147, 2005.

[105] M.R. Mousavi, M.A. Reniers, and J.F. Groote. A syntactic commutativity format for SOS. *Information Processing Letters*, 93:217-223, 2005.

[106] R. De Nicola and M.C.B. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83-133, 1984.

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

[108] R. Paige and R.E. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973-989, 1987.

[109] J. Parrow. The expressive power of parallelism. *Future Generation of Computer Systems*, 6:271-285, 1990.

[110] D.M.R. Park. Concurrency and automata on infinite sequences. In *Proceedings of the 5th GI Conference*, volume 104 of LNCS, pages 167-183. Springer, 1981.

[111] I.C.C. Phillips and I. Ulidowski. Ordered SOS rules and weak bisimulation. In A. Edalat, S. Jourdan, and G. McCusker, editors, *Advances in Theory and Formal Methods of Computing*, pages 300-311. Imperial College Press, 1996.

[112] I.C.C. Phillips and I. Ulidowski. Reversing algebraic process calculi. In L. Aceto and A. Ingólfsdóttir, editors, (*Proceedings of the Conference on Foundations of Software Science and Computation Structures (FOSSACS'06)*, volume 3921 of LNCS, pages 246-260. Springer, 2006.

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

[114] G.D. Plotkin. A structural approach to operational semantics. *Journal of Logic and Algebraic Progamming (JLAP)*, 60:17-139, 2004. This article first appeared as [113].

[115] G.D. Plotkin. The origins of structural operational semantics. *Journal of Logic and Algebraic Programming (JLAP)*, 60:3-15, 2004.

[116] T.C. Przymusinski. The well-founded semantics coincides with the three-valued stable semantics. *Fundamenta Informaticae*, 13(4):445-463, 1990.

[117] M.A. Reniers, J.F. Groote, M.B. van der Zwaag, and J. van Wamel. Completeness of timed *μCRL. Fundamenta Informaticae*, 50(3-4):361-402, 2002.

[118] J.W. Sanders and P. Zuliani. Quantum programming. In R.C. Backhouse and J.N.F. de Oliveira, editors, *Mathematics of Program Construction (MPC'00)*, volume 1837 of LNCS, pages 8099. Springer, 2000.

[119] A. Sabelfeld and A.C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1):5-19, 2003.

[120] D. Sands. From SOS rules to proof principles: An operational metatheory for functional languages. In *Proceedings of the 24th ACM Symposium on Principles of Programming Languages (POPL'97)*, pages 428-441. ACM, 1997.

[121] D. Sangiorgi. Bisimulation for higher-order process calculi. *Information and Computation*, 131(2):141-178, 1996.

[122] D. Sangiorgi, and D. Walker. *π-Calculus: A theory of mobile processes.* Cambridge University Press, 2001.

[123] A. Simpson. Sequent calculi for process verification: Hennessy-Milner logic for an arbitrary GSOS. *Journal of Logic and Algebraic Programming*, 60-61:287-322, 2004.

[124] A. Sokolova and E.P. de Vink. Probabilistic automata: System types, parallel composition and comparison. In C. Baier, B.R. Haverkort, H. Hermanns, J.-P. Katoen, and M. Siegle, editors, *Validation of Stochastic Systems - A Guide to Current Research*, volume 2925 of LNCS, pages 1-43. Springer, 2004.

[125] B. Thomsen. A theory of higher order communicating systems. *Information and Computation*, 116:38-57, 1995.

[126] S. Tini. Rule formats for non interference. In P. Degano, editor, *Proceedings of the 12th European Symposium on Programming, Programming Languages and Systems (ESOP'03)*, volume 2618 of LNCS, pages 129-143. Springer, 2003.

[127] S. Tini. Rule formats for compositional non-interference properties. *Journal of Logic and Algebraic Progamming*, 60:353-400, 2004.

[128] D. Turi. *Functorial Operational Semantics and its Denotational Dual.* PhD Thesis, Vrije University Amsterdam, 1996.

[129] D.A. Turner. Miranda: a non-strict functional language with polymorphic types. In J.-P. Jouannaud, editor, *Proceeding of the ACM Conference on Functional Programming Languages and Computer Architecture*, volume 201 of LNCS, pages 1-16. Springer, 1985.

[130] I. Ulidowski. Equivalences on observable processes. In *Proceedings of the 7th IEEE Symposium on Logic in Computer Science (LICS'92)*, pages 148-159. IEEE CS, 1992.

[131] I. Ulidowski. Axiomatisations of weak equivalences for De Simone languages. Extended abstract. In I. Lee and S.A. Smolka, editors, *Proceedings of the 6th Conference on Concurrency Theory (CONCUR'95)*, volume 962 of LNCS, pages 219-233. Springer 1995.

[132] I. Ulidowski. Finite axiom systems for testing preorder and De Simone process languages. *Theoretical Computer Science*, 239(1):97-139, 2000.

[133] I. Ulidowski. Rewrite Systems for OSOS Process languages. In R. Amadio and D. Lugiez, editors, *Proceedings of the 14th International Conference on Concurrency Theory (CONCUR'03)*, volume 2761 of LNCS, pages 87-102. Springer, 2003.

[134] I. Ulidowski and I.C.C. Phillips. Formats of ordered SOS rules with silent actions. In M. Bidoit and M. Dauchet, editors, *Proceedings of 7th International Joint Conference on Theory and Practice of Software Development (TAPSOFT'97)*, volume 1214 of LNCS, pages 297-308. Springer, 1997.

[135] I. Ulidowski and I.C.C. Phillips. Ordered SOS rules and process languages for branching and eager bisimulations. *Information and Computation*, 178(1):180-213, 2002.

[136] I. Ulidowski and S. Yuen. Process languages with discrete time based on the Ordered SOS format and rooted eager bisimulation. *Journal of Logic and Algebraic Programming*, 60-61:401-461, 2004.

[137] F.W. Vaandrager. Expressiveness results for process algebras. In J.W. de Bakker, W.P. de Roever, and G. Rozenberg, editors, *Proceedings of the REX Workshop on Semantics*, volume 666 of LNCS, pages 609-638. Springer, 1993.

[138] F.W. Vaandrager. On the relationship between process algebra and input/output automata. In *Proceedings of the 6th Annual IEEE Symposium on Logic in Computer Science (LICS'91)*, pages 387-398. IEEE CS, 1991.

[139] A. Verdejo and N. Martí-Oliet. Implementing CCS in Maude. In T. Bolognesi and D. Latella, editors, *Proceedings of the IFIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XIII) and Protocol Specification, Testing and Verification (PSTV XX)*, volume 183 of *IFIP Conference Proceedings*, pages 351-366. Kluwer, 2000.

[140] A. Verdejo and N. Martí-Oliet. Implementing CCS in Maude 2. In F. Gadducci and U. Montanari, editors, *Proceedings of the 4th International Workshop on Rewriting Logic and its Applications (WRLA'02)*, volume 71 of ENTCS, pages 239-257. Elsevier, 2002.

[141] A. Verdejo. Building tools for LOTOS symbolic semantics in Maude. In D. Peled and M. Vardi, editors, *Proceedings of the 22nd IFIP International Conference on Formal Techniques for Networked and Distributed Systenms (FORTE'02)*, volume 2529 of LNCS, pages 292-307. Springer, 2002.

[142] A. Verdejo and N. Martí-Oliet. Two Case Studies of Semantics Execution in Maude: CCS and LOTOS. *Formal Methods in System Design* 27(1-2):113-172, 2005.

[143] C. Verhoef. A general conservative extension theorem in process algebra. In E.-R. Olderog, editor, *Proceedings of 3rd IFIP Working Conference on Programming Concepts, Methods and Calculi (PROCOMET'94)*, volume A-56 of *IFIP Transactions*, pages 274-302. Elsevier, 1994.

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

[145] C. Verhoef, L. Aceto, and W.J. Fokkink. Conservative extension in structural operational semantics. *Bulletin of the European Association for Theoretical Computer Science (EATCS)*, 69:110-132, 1999.

[146] S. Weber and B. Bloom. Metatheory of the π-calculus. Technical Report TR96-1564, Department of Computer Science, Cornell University, Ithaka, NY, USA, 1996.

[147] A. Ziegler, D. Miller, and C. Palamidessi. A Congruence Format for Name-Passing Calculi. In P.D. Mosses and I. Ulidowski, editors, *Proceedings of the 2nd Workshop on Structural Operational Semantics (SOS 2005)*, volume 156(1) of ENTCS, pages 169-189. Elsevier, 2005.