A Hierarchy of SOS Rule Formats

Jan Friso Groote [1] MohammadReza Mousavi [2]
Michel A. Reniers [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 [62]. 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- and probability-related issues. In this paper, we provide an initial hierarchy of SOS rules formats and meta-theorems formulated around them.

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

1 Introduction

Structural Operational Semantics has become the common way to define operational semantics. Operational semantics defines the possible *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 constituting parts. This forms the central idea behind Structural Operational Semantics (SOS).

[1] Email: J.F.Groote@tue.nl
[2] Email: M.R.Mousavi@tue.nl
[3] Email: M.A.Reniers@tue.nl

Transition System Specifications (TSS's), as introduced by Groote and Vaandrager in [36], 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 [35,15,32] to security- [70,71] and probability-related issues [10,41]. 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*.

The excellent overview [3] provides existing rule formats to its date of publication (2001). Since then, even more formats have been proposed and we felt that in order to keep track, this field of formats requires some structure. Therefore, we attempt to present an overview of all SOS rule formats defined in the literature, together with the meta-theorems formulated around them. All the results are put in a lattice to easily locate the most suited format for a certain application. To do this, we define the concept of a TSS, in a far more general setting than [36], including the concepts of multi-sorted signatures and variable binding, inspired by the definition of [24]. This general definition of TSS serves 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 the hierarchy of formats is given. In Section 3, we list different syntactic features that an SOS rules can have. In Section 4, we review semantic meta-theorems about different SOS frameworks.

Disclaimer. This paper is a step towards a complete survey. This means that although most formats and results are mentioned in the remainder of this paper, some rule formats and meta-result can still be missing. We will be very thankful if notified of such omissions.

Acknowledgements. Peter Mosses, Simone Tini and Irek Ulidowski provided useful comments on the draft of this article.

2 A hierarchy of operational formats

Since the first formats have been defined for TSS rules, many 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 (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. The one inclusion that is not syntactic but possibly require some translation of syntactic constructs is indicated by a dotted line.

A node in this lattice has the structure shown in the left-hand-side below.

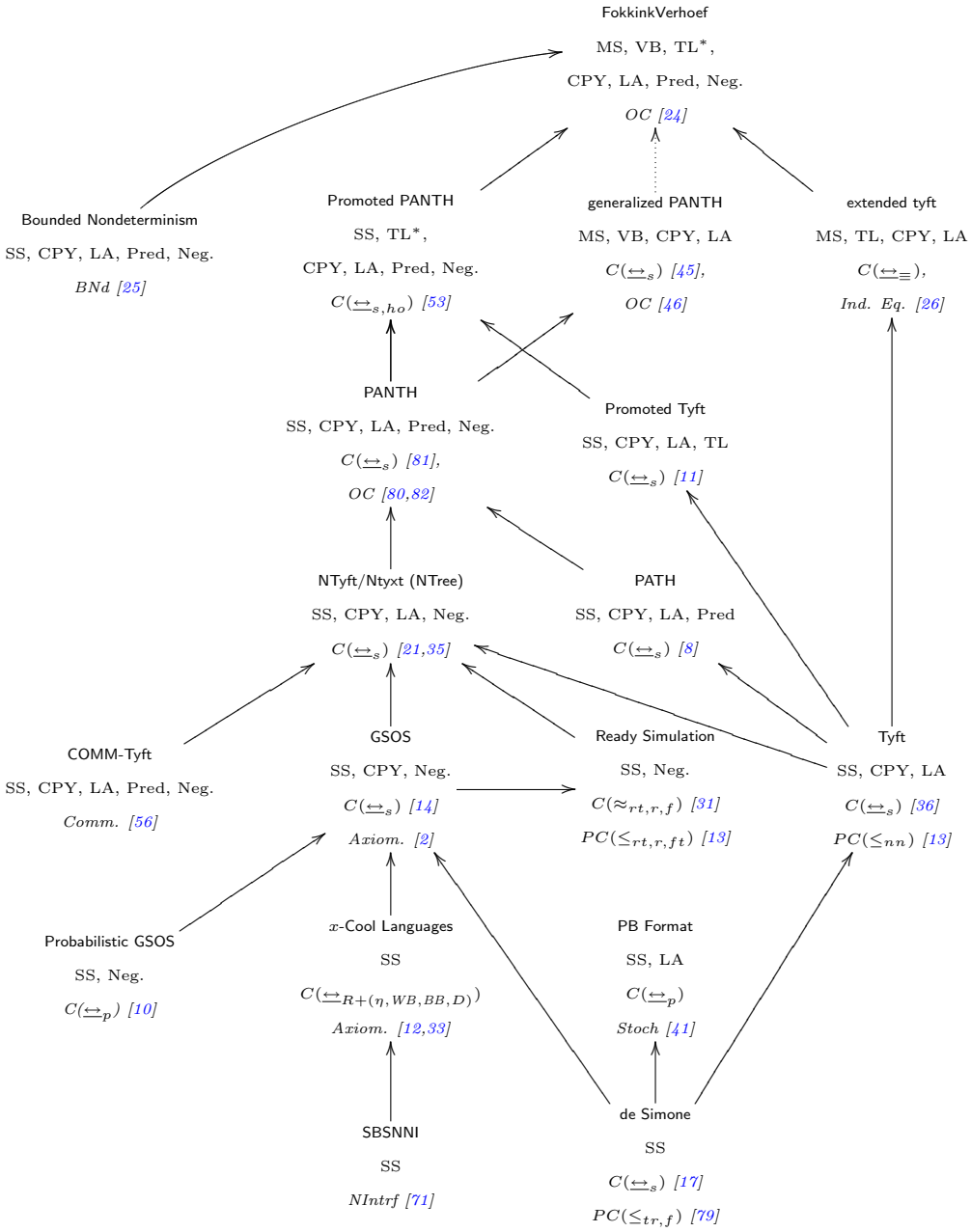

Fig. 1. A hierarchy of existing SOS formats

At the right an example is shown, taken from the lattice.

Format Name	NTyft/NTyxt (NTree)
Syntactic Features	SS, CPY, LA, Neg.
Semantic Meta-Theorems [references]	$C(\underline{\leftrightarrow}_s)$ *[21,35]*

The syntactic features of each format are described in Section 3. The abbreviations can be found in Table 1. The theorems holding for TSS's in

each format are described in Section 4 and the abbreviations can be found in Table 2.

3 Syntactic Features of TSS's

In this section we list the distinctive syntactical features of the different rule formats. The typical features for each format are listed in Table 1.

Shorthand	**Syntactic Feature**
SS vs. MS	Single- vs. Multi-Sorted Terms
VB	Variable Binding
TL vs. TL*	A Term vs. List of Terms as Labels
CPY	Copying Variables
LA	Lookahead
Pred	Predicates
Neg	Negative Premises

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

3.1 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.

Open Terms as Labels. Many SOS frameworks assume a special sort for labels and only allow for constants (alternatively, closed terms) of this sort to appear as labels. Such SOS frameworks thus forbid any correlation between valuation of terms and labels through the use of common variables. Frameworks defined in [24,26,11,53] allow for arbitrary terms as labels. All other SOS frameworks reviewed in this paper only allow for constant labels.

Open terms are used as labels in a number of cases in transition system specifications. Higher-order process calculi [16,69,66] are examples of formalisms exploiting this feature.

Two other frameworks that use labels with some structure on them are the Modular SOS framework of [51,52] and Enhanced Operational Semantics of [18]. 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.

Lists of Terms as Labels. Most SOS frameworks only allow for a single term as label. The only existing exceptions are those of [24,53].

3.2 Signatures

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 [48,49,50], 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, [48], [44,63] and [7], respectively). Hence, it is interesting to accommodate the concept of names in the TSS framework.

There have been a few attempts in this direction. Proposals for modeling names and binders are formulated in [83,24,45,46,39,65]. Apart from the introduction of parameterized variables, the TSS frameworks of [83,45,46,39,65] are more restricted than that of [24] in that they do not allow for open terms as labels. Furthermore, the frameworks of [83,39,65] do not support negative premises.

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

Multi-Sorted States. Based on the number of sorts allowed in the signature, an SOS framework may be classified in the following three categories:

(i) Multi-sorted TSS's: In such frameworks, there is no restriction on the sorts allowed for constructing terms.

(ii) 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 [55] 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.

(iii) Single-sorted: This is the most common framework in the literature. 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.

The TSS of [26] 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.

3.3 Positive Premises and the Conclusion

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 [28] 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.

Well-foundedness. 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 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 [72]) . A simple example of copying is the second rule in the following TSS which defines the semantics of the `while` construct.

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

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

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 following example from [58] illustrates a possible use of deduction rules with infinite premises:

$$\frac{x \xrightarrow{a} y \quad a \notin H}{\partial_H(x) \xrightarrow{a} \partial_H(x')} \qquad \frac{\forall_{a \in A \setminus H} x \xrightarrow{a} \!\!\!\!\!\rightarrow}{\partial_H(x) \xrightarrow{\chi} \delta}$$

The above deduction rules define the semantics of the encapsulation operator $\partial_H(\_)$ which forbids its parameter from performing actions 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.

3.4 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 [6] in the specification of the semantics of the following priority operator $\theta(\_)$.

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

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 ordering $>$). In addition to negative premises, the above deduction rule may have infinite premises if there is a chain of priorities with infinitely many (different) basic actions.

3.5 Predicates

Predicates are useful syntactic features which are used to specify phenomena such as termination or divergence.

3.6 Other Syntactic Features

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 defined in Section 3.4 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 as follows.

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

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 part 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 [74,75,76,61].

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

4 Semantic Meta-Results

In this section we list a number of meta-results. Each meta-result is abbreviated (see Table 2) and indicated in each format in Figure 1. To fix the notation for the semantic properties and their corresponding meta-result, next, we define the notion of TSS.

Definition 4.1 (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:*

$$\frac{\{P(L_i)t_i \quad or \quad t_i \xrightarrow{L_i} t'_i \mid i \in I\} \quad \{\neg P(L_j)t_j \quad or \quad t_j \xrightarrow{L_j} \mid j \in J\}}{P(L)t \quad or \quad t \xrightarrow{L} t'}$$

In the above rule schema t, t_x and t'_x stand for terms from signature Σ and

L and L_x are lists of terms from the same signature. Statements of the form $P(L)t$ and $\neg P(L)t$ are called positive and negative predicate formulae, respectively. Statements of the form $t \xrightarrow{L} t'$ and $t \xrightarrow{L} \not\rightarrow$ (where in the latter statement t' may or may not be present) are called positive and negative transition formulae, respectively. The sets I and J are arbitrary (possibly infinite) index sets. In a deduction rule, the statements above the line are called premises and the one below the line is called conclusion.

For TSS's with constant labels, a distinguished sort L is dedicated to represent the labels and then the lists L and L_x in the above rule are confined to have a single constant (closed term) from this sort. In the literature, as well as in the remainder of this paper, for TSS's with constant labels, sort L is taken out of the signature Σ and such TSS's are then represented by a triple (Σ, L, D).

Semantic Meta-Theorems

Shorthand	**Semantic Meta-Theorems**
$C(\underline{\leftrightarrow}_x)$	Congruence for x-Bisimulation
$C(\approx_x)$	Congruence for x-equality
$PC(\leq_x)$	Pre-congruence for x-pre-order
OC	Operational Conservativity
Axiom.	Deriving Sound and Complete Axiomatization
Ind. Eq.	Comparison of Induced Equality Classes
NIntrf	Non Intereference (Security-related [64])
BNd	Bounded Non-determinism
Stoch	Stochasticity

Table 2
Short-hands for theorems used in Figure 1

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 literature [30,29]. Correspondingly, there are a number of rule formats guaranteeing these notions to be a (pre-)congruence [36,13,12,33]. 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 in the left-hand side of the arrow). The only congruence meta-theorems for multi-sorted frameworks are those of [26,45,24,83] and with open terms as labels are [26,24,53].

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

Definition 4.2 ((Pre-)Congruence) *An equivalence (pre-order) $R \subseteq \mathcal{C} \times \mathcal{C}$ is a (pre-)congruence with respect to a signature Σ if and only if for all $(f, ar(f)) \in \Sigma$ and all $\overrightarrow{p}_{ar(f)}, \overrightarrow{q}_{ar(f)} \in \mathcal{T}$, if $\overrightarrow{p}_{ar(f)}\ R\ \overrightarrow{q}_{ar(f)}$ then $f(\overrightarrow{p}_{ar(f)})\ R\ f(\overrightarrow{q}_{ar(f)})$.*

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

Definition 4.3 (Bisimulation and Bisimilarity [60]) *A relation $R \subseteq \mathcal{C} \times \mathcal{C}$ is a bisimulation relation with respect to a set of transition relations Rel and a set of predicates Pr if and only if $\forall_{p,q \in \mathcal{C}}\ pRq \Rightarrow \forall_{r \in Rel, P \in Pr}$ such that r and P are of arbitrary arities of n and m,*

(i) $\forall_{p' \in \mathcal{C}, l \in L}\ p \xrightarrow{l} p' \Rightarrow \exists_{q' \in \mathcal{C}}\ q \xrightarrow{l} q' \wedge (p', q') \in R$;

(ii) $\forall_{q' \in \mathcal{C}, l \in L}\ q \xrightarrow{l} q' \Rightarrow \exists_{p' \in \mathcal{C}}\ p \xrightarrow{l} p' \wedge (p', q') \in R$;

(iii) $P(l)\ p \Leftrightarrow P(l)\ q$.

Two closed terms p and q are are bisimilar if and only if there exists a bisimulation relation R with respect to Rel such that $(p, q) \in R$. Two closed terms p and q are bisimilar with respect to a transition system specification tss, denoted by $tss \vdash p \leftrightarrows q$, 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.

(i) 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;

(ii) For finite state processes, strong bisimilarity can be checked very efficiently in practice [59] while some weaker notions are intractable [40];

(iii) Other notions can often be coded in strong bisimilarity [28].

So, it is not surprising that the first standard congruence format was geared toward strong bisimilarity. This format was proposed by De Simone in [17].

Definition 4.4 (De Simone Format) *The De Simone format uses the positive framework with constant labels and allows for deduction rules of the following form:*

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

where x_i and y_i are distinct variables ranging over process terms, f is a function symbol from the signature (e.g., sequential composition, parallel composition, etc.), I is a subset of the set $\{1, \ldots, ar(f)\}$ (indices of arguments of f), 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 Pred is a predicate stating the relationship between the labels of the premises and the label of the conclusion.

Bloom, Istrail and Meyer, in their study of the relationship between bisimilarity and completed-trace congruence [14], define an extension of the De Simone format, called GSOS (for Structural Operational Semantics with Guarded recursion), to capture *reasonable* language definitions. The GSOS format extends the De Simone format by allowing for copying and negative premises. The GSOS format is formally defined as follows.

Definition 4.5 (GSOS Format) *A deduction rule is in the GSOS format when it is of the following form:*

$$\frac{\{x_i \xrightarrow{l_{ij}} y_{ij} \mid i \in I, 0 \leq j \leq m_i\} \cup \{x_j \overset{l_{jk}}{\nrightarrow} \mid j \in J, 0 \leq k \leq n_j\}}{f(\overrightarrow{x}_{ar(f)}) \xrightarrow{l} t}.$$

where f is a function symbol, x_i $(1 \leq i \leq ar(f))$ and y_{ij}'s ($i \in I$ and $j \leq m_i$) 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), $vars(t) \subseteq \{x_i, y_{jk} | i \in I \cup J, j \in I, k \leq m_i\}$ and l_{ij}'s, l_{jk}'s and l are constant labels. A TSS is in the GSOS format when all its deduction rules are.

Another orthogonal extension of the De Simone format is called tyft/tyxt format and is first formulated in [36].[4] This format allows for lookahead,

[4] Tyft/tyxt is a code representing the structure of symbols in the deduction rules, namely,

copying and an infinite set of premises.

Definition 4.6 (Tyft/tyxt Format [36]) *A rule is in the tyft format if and only if it has the following form.*

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

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 a function symbol from the signature, I is a (possibly infinite) set of indices, t and t_i's are arbitrary terms and 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 conclusion is a variable distinct from all targets of 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 [36], to prove congruence of strong bisimilarity for TSS's in the tyft/tyxt format, well-foundedness of the TSS is assumed. Later, in [19], 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.

Theorem 4.7 (Congruence of Bisimilarity for Tyft/tyxt [36,19]) *For a TSS in the tyft/tyxt format, strong bisimilarity is a congruence.*

The merits of the two extensions were merged in [35] where negative premises were added to the tyft/tyxt format, resulting in the ntyft/ntyxt format.

Definition 4.8 (Ntyft/ntyxt Format [35]) *A rule is in the ntyft format if and only if it has the following form.*

$$\frac{\{t_i \xrightarrow{l_i} y_i | i \in I\} \quad \{t_j \xrightarrow{l_j} | j \in J\}}{f(\overrightarrow{x}_{ar(f)}) \xrightarrow{l} t}$$

The same conditions as of the tyft format hold for the positive premises and the conclusion. 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

---

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.

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 the coming section, we give well-definedness criteria (from [35,15]) for the semantics of TSS's in the ntyft/ntyxt format. Well-foundedness assumption was also used in [35] and was shown to be redundant in [21].

Finally, the PATH format [8] for Predicates And Tyft/tyxt Hybrid format) and the PANTH format [81] (for Predicates And Negative Tyft/tyxt Hybrid format) extend the tyft/tyxt and the ntyft/ntyxt with predicates, respectively.

Definition 4.9 (PANTH Format [35]) *A rule is in the PANTH format if and only if it has the following form.*

$$\frac{\{P_i(l_i)\ t_i\ \textit{or}\ t_i \xrightarrow{l_i} y_i | i \in I\} \quad \{\neg P(l_j)\ t_j\ \textit{or}\ t_j \xrightarrow{l_j} | j \in J\}}{P(l)\ t\ \textit{or}\ f(\overrightarrow{x}_{ar(f)}) \xrightarrow{l} t\ \textit{or}\ x \xrightarrow{l} t}$$

The same conditions as of 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).

[11,53] generalize tyft/tyxt and PANTH format, respectively, to their promoted-counterparts which cater for open terms as labels.

In [45], the PANTH format is extended for multi-sorted variable binding. This covers the problem of operators such as recursion or choice over a time domain. The issue of binding operators for multi-sorted process terms is also briefly introduced in [3].

A number of other standard formats have been proposed for the congruence of weaker notions of bisimulation. A major example of such formats is the Cool languages format introduced in [12] which proves congruence of (rooted) weak and branching bisimulations. This format has recently been reformulated in [33] and extended to prove congruence of delay and η-bisimulations. In [31], 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 [13] to obtain pre-congruence for readiness, ready traces and failures pre-order. Note that pre-congruence for a pre-order implies congruence for the corresponding equivalence (the kernel of the pre-order). Fokkink in [20] presents the RBB Safe format which induces congruence for rooted branching bisimulation and generalizes the Cool languages format of [12] to the setting with negative premises and predicates.

In [53], the higher-order PANTH format is presented which induces congruence for higher-order bisimilarity [4]. The rule format of [83] guarantees that open-bisimilarity [67] is a congruence.

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:

$$\frac{c \overset{a}{\not\rightarrow}}{c \overset{a}{\rightarrow} c} \quad \Bigg| \quad \frac{c \overset{a}{\not\rightarrow}}{c \overset{b}{\rightarrow} c} \qquad \frac{c \overset{b}{\not\rightarrow}}{c \overset{a}{\rightarrow} c}$$

The left-hand-side TSS is kind of paradoxical and the right-hand-side one is ambiguous in that it is not clear why one would prefer one of the two possible transitions $c \overset{a}{\rightarrow} c$ or $c \overset{b}{\rightarrow} c$ over the other. ([32] gives an overview of this issue, present 11 different semantic interpretations of TSS's and compares them formally.)

In [35], the first criterion is given using which a TSS in the ntyft/ntyxt format is guaranteed to have a well-defined semantics. This criterion, defined below, is called (strict) stratification and is originally due to [27] in logic programming. It is an important property of a format when it guarantees that every set of rules equivocally defines a transition relation.

In [35], Groote defines a criterion which guarantees a TSS in the ntyft/ntyxt format to induce a well-defined semantics. This criterion, defined below, is called (strict) stratification and is originally due to [27] in the setting of logic programming.

Definition 4.10 (Stratification [35]) *A stratification of a transition system specification tss is a function $\mathcal{S}$ from closed positive formulae to an ordinal such that for all deduction rules in tss of the following form:*

$$\frac{\{t_i \overset{l_i}{\rightarrow} t_i' | i \in I\} \quad \{t_j \overset{l_j}{\not\rightarrow} | j \in J\}}{f(\overrightarrow{x}_{ar(f)}) \overset{l}{\rightarrow} t}$$

and for all closed substitutions σ, $\forall_{i \in I} \mathcal{S}(\sigma(t_i \overset{l_i}{\rightarrow} t_i')) \leq \mathcal{S}(\sigma(f(\overrightarrow{x}_{ar(f)}) \overset{l}{\rightarrow} t))$ and $\forall_{j \in J}\ \forall_{t' \in \mathcal{T}}\ \mathcal{S}(\sigma(t_j \overset{l_j}{\rightarrow} t')) < \mathcal{S}(\sigma(f(\overrightarrow{x}_{ar(f)}) \overset{l}{\rightarrow} t))$. A transition system specification 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.*

The following theorem shows useful properties of stratified TSS's.

Theorem 4.11 (Properties of Stratified TSS's [15]) *A stratified TSS in the ntyft/ntyxt format has a unique semantics (called its stable model) and for such a TSS bisimilarity is a congruence.*

TSS's in the GSOS format are strictly stratified using a measure of size on terms in the source of the transition formulae. As a corollary of the above theorem, one may deduce that the semantics of a TSS in the GSOS format is well-defined and bisimilarity is a congruence for such a TSS. Of course, congruence of bisimilarity for the GSOS format had been directly proven in [14].

Definition 4.10 and Theorem 4.11 can easily be extended to the PANTH format (by interpreting predicates as transitions with dummy targets). Theorem 4.11 has been generalized to the so-called, *positive after reduction* or *complete TSS's* in [15]. In the same paper, an of a TSS in the ntyft/ntyxt format is given for which bisimilarity is *not* a congruence. This TSS, does not satisfy the criterion of being positive after reduction (and thus, not stratified), yet has a unique semantics.

4.3 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. Two general instances of such meta-theorems are formulated in [24,46]. We review the results of [24] (simplified to the setting with constant labels and without binders) in this section, which gives the most detailed account of this issue.

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 4.12 (Extension of a TSS) *Consider TSS's* $tss_0 = (\Sigma_0, L_0, D_0)$ *and* $tss_1 = (\Sigma_1, L_1, D_1)$. *The extension of* tss_0 *with* tss_1, *denoted by* $tss_0 \oplus tss_1$, *is defined as* $(\Sigma_0 \cup \Sigma_1, L_0 \cup L_1, D_0 \cup D_1)$.

Next, we define when an extension of a TSS is called operationally conservative.

Definition 4.13 (Operational Conservativity [80]) *Consider TSS's* $tss_0 = (\Sigma_0, L_0, D_0)$ *and* $tss_1 = (\Sigma_1, L_1, D_1)$. *If* $\forall_{p \in C(\Sigma_0)}\ \forall_{p' \in C(\Sigma_0 \cup \Sigma_1)}\ \forall_{l \in L_0 \cup L_1}$ $tss_0 \cup tss_1 \vDash p \xrightarrow{l} p' \Leftrightarrow tss_0 \vDash p \xrightarrow{l} p'$ *(and similarly,* $tss_0 \cup tss_1 \vDash P(l)\ p \Leftrightarrow tss_0 \vDash P(l)\ p$*) , then* $tss_0 \cup tss_1$ *is an operationally conservative extension of* tss_0.

Next, we formulate sufficient conditions to prove operational conservativity. But before that, we need a few auxiliary definitions.

Definition 4.14 (Source Dependency) *All variables appearing in the source of the conclusion of a deduction rule are called* source dependent*. A variable of a deduction rule is* source dependent *if it appears in a target of a premise of which all the variables of the source are source dependent. A premise is* source dependent *when all the variables appearing in it are source dependent. A rule is* source dependent *when all its variables are. A TSS is* source dependent *when all its rules are.*

Definition 4.15 (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.*

The following result, from [24], gives sufficient conditions for an extension of a TSS to be operationally conservative.

Theorem 4.16 (Operational Conservativity Meta-Theorem [24]) *Given two TSS's $tss_0 = (\Sigma_0, L_0, D_0)$ and $tss_1 = (\Sigma_1, L_1, D_1)$, $tss_0 \cup tss_1$ is an operationally conservative extension of tss_0 if:*

(i) *tss_0 is source dependent;*

(ii) *for all $d \in D_1$ at least one of the following holds:*
  (a) *the source of the conclusion has a function symbol in $\Sigma_1 \setminus \Sigma_0$, or*
  (b) *$\rho(d, \Sigma_0)$ has a source-dependent positive premise $t \xrightarrow{l} t'$ such that $l \notin \Sigma_0$ or $t' \notin T(\Sigma_0)$.*

In [57], slightly more liberal notion of operational conservativity, called *orthogonality*, is introduced and a meta-theorem about it is proposed. 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.

4.4 Generating Equational Theories

Equational theories are central notions to process algebras [5,38,47]. They capture the basic intuition behind the algebra, and the models of the algebra are expected to respect this intuition (e.g., the models induced by operational semantics modulo bisimilarity). One of the added-values 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*, phrases that all induced behavioral equalities are derivable from the equational theory, as well. These concepts are formalized in the remainder.

Definition 4.17 (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 $p, p' \in \mathcal{C}$, 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 $p, p' \in \mathcal{C}$, 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 [2,1], an automatic method for generating sound and complete equational theories from GSOS specifications is presented. This technique was extended in [9] 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 [2]. Axiom systems for pre-orders have been generated, too, see for instance [73]. Along the same lines, [74] generates prioritized rewrite systems for TSS's with an ordering on deduction rules.

4.5 Other Meta-Results

Non-Interference. Confidentiality is an important aspect of security and non-interference [34] 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 [70,71] 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.

Decomposition of Logical Formulae. In [37], a logical framework, nowadays called the *Henessy-Milner logic* after the authors' names, is proposed. The Hennessy-Milner logic can be used to reason about processes and characterize their equalities. In [42,43] 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 [22] and extended to the ready simulation format (ntyft/ntyxt format without lookahead). In [23], the decomposition technique is used to derive congruence formats for the notion of eta-bisimulation, as an example. Simpson in [68] introduces a compositional proof system for checking Hennessy Milner formulae on processes specified in the GSOS format.

Stochasticity. For probabilistic transition systems, it is essential to make sure that the sum of all probabilities belonging to the same distribution amounts to 1 (or zero). This is called (semi-)stochasticity. In [41], a restricted form of the De Simone format is proposed that guarantees semi-stochasticity. To avoid dealing with negative premises the format of [41] supports ordering on rules.

Bounded Non-determinism. In [78], a rule format, by imposing restrictions on the De Simone format, is proposed 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 Duong Vu in [25] generalize the result of [78] to a far more general SOS framework.

Timing properties. In [77] a set of syntactic conditions have been defined that guarantee properties for timed systems, such as time determinism, persistence, maximal progress and patience.

References

[1] 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-Verlag, 1994.

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

[3] 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 Science, 2001.

[4] 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-Verlag, 1988.

[5] J.C.M. Baeten and J.A. Bergstra. Real Time Process Algebra. *Formal Aspects of Computing*, 3:142–188, 1991.

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

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

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

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

[10] 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 *Electronic Notes in Theoretical Computer Science*, pages 1–25, 2002.

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

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

[13] 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.

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

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

[16] 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 *Lecture Notes in Computer Science*, pages 149–161. Springer-Verlag, 1989.

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

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

[19] 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-Verlag, 1994.

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

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

[22] W.J. Fokkink, R.J. van Glabbeek, and P. de Wind. Compositionality of Hennessy-Milner logic through structural operational semantics. In A.Lingas and B.J. Nilsson, editors, *Proceedings of the 14th Symposium on Fundamentals of Computation Theory (FCT'03)*, volume 2751 of *LNCS*, pages 412–422. Springer-Verlag, 2003.

[23] 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'05)*, Electronic Notes in Theoretical Computer Science, 2005.

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

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

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

[27] 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.

[28] 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 *Lecture Notes in Computer Science*, pages 336–347. Springer-Verlag, 1987.

[29] 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 Science, 2001.

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

[31] 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 Third International Conference on Algebraic Methodology and Software Technology (AMAST '93)*, pages 75–82. Springer-Verlag, 1993.

[32] 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.

[33] R.J. van Glabbeek. On cool congruence formats for weak bisimulations (extended abstract). In D.V. Hung and M. Wirsing, editors, *Proceedings International Colloquium on Theoretical Aspects of Computing (ICTAC05)*, volume 3722 of LNCS, pp. 331-346. Springer-Verlag, 2005.

[34] J.A. Goguen and J. Meseguer. Security policies and security models. In *IEEE Symposium on Security and Privacy*, pages 11–20. IEEE Computer Society Press, 1982.

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

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

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

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

[39] D.J. Howe. Proving Congruence of Bisimulation in Functional Programming Languages. *Information and Computation*, 124(2):103–112, 1996.

[40] 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.

[41] 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 *Lecture Notes in Computer Science*, pages 63–78. Springer-Verlag, 2005.

[42] K.G. Larsen. *Context-Dependent Bisimulation between Processes*. PhD thesis, University of Edinburgh, 1986.

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

[44] S.P. Luttik. *Choice quantification in process algebra*. PhD thesis, Department of Computer Science, University of Amsterdam, 2002.

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

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

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

[48] R. Milner. The polyadic π-calculus: a tutorial. In FriedrichL. Bauer, Wilfried Brauer, and Helmut Schwichtenberg, editors, *Logic and Algebra of Specification*, pages 203–246. Springer-Verlag, 1993.

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

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

[51] P.D. Mosses, *Exploiting Labels in Structural Operational Semantics*. Fundamenta Informaticae, 60(1-4):17–31, 2004.

[52] P.D. Mosses, *Modular Structural Operational Semantics*. *Journal of Logic and Algebraic Programming*, 60-61:195–228, 2004.

[53] 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, pp. 308–322. Springer-Verlag, 2005.

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

[55] 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.

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

[57] 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 *Lecture Notes in Computer Science*, pages 1214–1225. Springer-Verlag, 2005.

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

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

[60] 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-Verlag, 1981.

[61] I.C.C. Phillips and I. Ulidowski. Ordered SOS rules and weak bisimulation. In Abbas Edalat, Sofia Jourdan, and Guy McCusker, editors, *Advances in Theory and Formal Methods of Computing*, pages 300–311. Imperial College Press, London, UK, 1996.

[62] G.D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, Aarhus, Denmark, September 1981. Also published in: *Journal of Logic and Algebraic Programming* 60-61:17–140, 2004.

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

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

[65] 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)*, pp. 428–441, ACM Press, 1997.

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

[67] D. Sangiorgi, and D. Walker. π-Calculus: A Theory of Mobile Processes. Cambridge University Press, 2001.

[68] 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.

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

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

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

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

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

[74] 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, pp. 87-102. Springer–Verlag, 2003.

[75] 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-Verlag, 1997.

[76] 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.

[77] 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.

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

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

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

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

[82] 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.

[83] 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'05)*, Electronic Notes in Computer Science. Elsevier Science, 2005.