The Concurrency Column

BY

Luca Aceto

BRICS, Department of Computer Science
Aalborg University, 9220 Aalborg Ø, Denmark

Dept. of Computer Science, School of Science and Engineering
Reykjavik University, 103 Reykjavik, Iceland

luca@{cs.auc.dk,ru.is}, http://www.cs.auc.dk/~luca/BEATCS

In light of its flexibility and its intuitive appeal, Structural Operational Semantics is by now one of the most popular approaches to giving the semantics of programming and specification languages. Over the last twenty five years, researchers in concurrency theory have developed a number of general results guaranteeing semantic properties for whole classes of languages, provided their operational semantics is given by rules that meet certain syntactic constraints. Some of these results link operational semantics with algebraic semantics in that they aim either at guaranteeing the validity of general algebraic properties of language constructs "by design" or at generating valid algebraic equations from the operational specification of language constructs. This contribution to the Concurrency Column offers a survey of some of the methods that have been developed in the literature on this line of research. It provides a glimpse of an ongoing research programme, which I hope will be of interest to the readers of this column.

On a different note, I am happy to report that CONCUR 2009 in Bologna was a resounding success. The scientific programme for the main conference and its affiliated events was of very high quality and the organization was outstanding. Many thanks to Mario Bravetti, Gianluigi Zavattaro and their team for organizing a splendid twentieth edition of CONCUR. I encourage the readers of the Concurrency Column to read the proceedings of CONCUR 2009. There is plenty of food for thought on those pages and this bodes well for the future of concurrency theory and of the CONCUR conference series.

Algebraic Properties for Free!*

Luca Aceto, Anna Ingolfsdottir
School of Computer Science, Reykjavik University,
Kringlan 1, IS-103, Reykjavik, Iceland

MohammadReza Mousavi, Michel A. Reniers
Department of Computer Science,
Eindhoven University of Technology,
P.O. Box 513, NL-5600 MB Eindhoven, The Netherlands

Abstract

Algebraic properties specify some natural properties of programming and specification constructs. This paper provides an overview of techniques to guarantee or generate algebraic properties of language constructs by investigating the syntactic shape of the deduction rules defining their operational semantics.

1 Introduction

Programming and specification languages are defined in terms of a syntax and a semantics. The syntax of a language specifies the grammatical structure of well-formed programs, while its semantics defines the intended meaning of syntactically valid programs. The definition of the syntax of a language is usually given in terms of a grammar, most often in the Backus-Naur Form or one of its variants. The syntax of a language can also be seen as an algebraic structure with programming or specification constructs as operators, or function symbols, that allow one to construct composite program fragments from their components. This paves the way to the use of algebraic methods for reasoning about programs. In fact, when designing a language, a designer usually has certain algebraic properties of operators in mind, e.g., $x_0; \mathit{skip} = x_0$. Such properties can either be validated

*The work of Aceto and Ingolfsdottir has been partially supported by the projects "The Equational Logic of Parallel Processes" (nr. 060013021) and "New Developments in Operational Semantics" (nr. 080039021) of the Icelandic Research Fund.

using the semantics of the language with respect to a suitable notion of program equivalence, or they can be guaranteed a priori "by design" [4, 13, 39, 31, 18, 3].

Algebraic properties specify some natural properties of function symbols from a given signature. Examples of such properties include commutativity, associativity and idempotence of binary operators, which are concisely specified, respectively, by the following equations:

$$f(x_0, x_1) = f(x_1, x_0) \qquad f(f(x_0, x_1), x_2) = f(x_0, f(x_1, x_2)) \qquad f(x_0, x_0) = x_0.$$

In the context of programming and specification languages, algebraic properties prescribe the equivalences that must be respected by all the semantic models of a language. In other words, they ensure that the semantics of certain composition schemes must preserve the "intended behaviour" of programs. The most popular way to define the operational semantics (hence the "behaviour") of computer languages is Structural Operational Semantics (SOS) [37, 36, 24]. Hence, it makes sense to establish generic methods that can guarantee and/or derive algebraic properties based on the SOS specifications of language constructs. This paper provides an overview of some of the available techniques to this end. In keeping with the expository nature of this article, we do not present the results in full generality since this would obscure the main message. However, we provide references to the literature for the readers interested in the technical details and generalizations of the results we cover in this survey.

The rest of this paper is organized as follows. In Section 2, we present some standard notions from the meta-theory of SOS. In Sections 3, 4 and 5, respectively, we report on work that aims at isolating sufficient syntactic conditions over SOS specifications that guarantee the validity of commutativity, associativity and idempotence axioms from SOS specifications. In Section 6, we present a survey of existing meta-theorems to generate sound and ground-complete axiom systems from SOS specifications. Since these axiom systems are ground-complete, all ground instances of abstract algebraic properties such as commutativity, associativity and idempotence can be derived from them. However, the generic meta-theorems presented in Section 6 do not derive such axioms explicitly. Hence, the techniques presented in Sections 3-5 can be complementary to those presented in Section 6. We get back to this issue in our concluding remarks and open problems, which are presented in Section 7.

2 Preliminaries

In this section we review, for the sake of completeness, some standard definitions from the meta-theory of SOS that will be used in the remainder of the paper. We refer the interested reader to [7, 32] for further details.

Definition 1 (Signature and terms)**.** *We let V represent an infinite set of variables with typical members $x, x', x_i, y, y', y_i, \ldots$. A* signature Σ *is a set of function symbols, each with a fixed arity. We call these symbols* operators *and usually represent them by $f, g, \ldots$. An operator with arity zero is called a* constant*. We define the set $\mathbb{T}(\Sigma)$ of* terms *over Σ as the smallest set satisfying the following constraints.*

- *A variable $x \in V$ is a term.*
- *If $f \in \Sigma$ has arity n and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term.*

We use $t, t', t_i, \ldots$ to range over terms. We write $t_1 \equiv t_2$ if t_1 and t_2 are syntactically equal. The function $vars : \mathbb{T}(\Sigma) \to 2^V$ gives the set of variables appearing in a term. The set $\mathbb{C}(\Sigma) \subseteq \mathbb{T}(\Sigma)$ is the set of closed terms*, i.e., terms that contain no variables. We use $p, p', p_i, \ldots$ to range over closed terms. A* substitution σ *is a function of type $V \to \mathbb{T}(\Sigma)$. We extend the domain of substitutions to terms homomorphically. If the range of a substitution lies in $\mathbb{C}(\Sigma)$, we say that it is a* closed substitution*.*

Definition 2 (Transition System Specifications (TSS), formulae and transition systems)**.** *A* transition system specification *is a triplet (Σ, L, D) where*

- *Σ is a signature.*
- *L is a set of labels. If $l \in L$, and $t, t' \in \mathbb{T}(\Sigma)$ we say that $t \xrightarrow{l} t'$ is a* (positive) formula*. A formula is typically denoted by ϕ, ψ, ϕ', ϕ_i, ...*
- *D is a set of* deduction rules*, i.e., pairs of the form (Φ, ϕ) where Φ is a set of formulae and ϕ is a formula. We call the formulae contained in Φ the* premises *of the rule and ϕ the* conclusion*.*

We write $vars(r)$ to denote the set of variables appearing in a deduction rule r. We say that a formula is closed *if all of its terms are closed. Substitutions are also extended to formulae and sets of formulae in the natural way. A set of positive closed formulae is called a* transition system*. The transition system induced by a TSS is the set of provable positive formulae, using its deduction rules. We write $\mathcal{T} \vdash p \xrightarrow{l} p'$ if the transition $p \xrightarrow{l} p'$ is provable from the deduction rules in the TSS $\mathcal{T}$.*

We often refer to a formula $t \xrightarrow{l} t'$ as a *transition* with t being its *source*, l its *label*, and t' its *target*. Predicates over terms can be seen as transitions with a dummy target [41]. A deduction rule (Φ, ϕ) is typically written as $\frac{\Phi}{\phi}$. We call a deduction rule *f-defining* when the outermost function symbol appearing in the source of its conclusion is f. For simplicity, we assume in the remainder of this

paper that all deduction rules are f-defining for some function symbol f, i.e., there are no deduction rules with a variable as the source of its conclusion. Note that we have confined ourselves to positive TSSs, i.e., those that only use formulae of the form $t \xrightarrow{l} t'$; most of the results presented in what follows have already been extended to the setting with negative formulae, i.e., formulae of the form $t \overset{l}{\nrightarrow}$, as premises. We refer to the corresponding papers for further details.

To establish a link between the operational model and the algebraic properties, a notion of behavioural equivalence should be fixed. A very common notion of behavioural equivalence that is mentioned frequently in this paper is the following notion of strong bisimilarity [34, 28].

Definition 3 (Strong Bisimilarity)**.** *Let $\mathcal{T}$ be a TSS with signature Σ. A relation $R \subseteq \mathbb{C}(\Sigma) \times \mathbb{C}(\Sigma)$ is a* strong bisimulation relation *if and only if R is symmetric and for all $p_0, p_1, p_0' \in \mathbb{C}(\Sigma)$ and $l \in L$*

$$(p_0 \; R \; p_1 \wedge \mathcal{T} \vdash p_0 \xrightarrow{l} p_0') \Rightarrow \exists_{p_1' \in \mathbb{C}(\Sigma)} (\mathcal{T} \vdash p_1 \xrightarrow{l} p_1' \wedge p_0' \; R \; p_1').$$

Two terms $p_0, p_1 \in \mathbb{C}(\Sigma)$ are called strongly bisimilar*, denoted by $\mathcal{T} \vdash p_0 \leftrightarrow p_1$, when there exists a strong bisimulation relation R such that $p_0 \; R \; p_1$.*

Any equivalence relation $\sim$ over closed terms in a TSS $\mathcal{T}$ is extended to open terms in the standard fashion, i.e., for all $t_0, t_1 \in \mathbb{T}(\Sigma)$, the equation $t_0 = t_1$ holds over $\mathcal{T}$ modulo $\sim$ if, and only if, $\mathcal{T} \vdash \sigma(t_0) \sim \sigma(t_1)$ for each closed substitution σ.

Ideally, the notion of behavioural equivalence should coincide with the equational theory generated by a set of axioms describing the desired algebraic properties of the operators in a language. One side of this coincidence is captured by the *soundness* theorem, which states that all the (closed) equalities that are derivable from the axiom system using the rules of equational logic are indeed valid with respect to the particular notion of behavioural equivalence. The other side of the coincidence, called *ground-completeness*, states that all the valid behavioural equivalences (on closed terms) are derivable from the axiom system, as well. These concepts are formalized in what follows.

Definition 4 (Axiom System)**.** *An* axiom system *E over a signature Σ is a set of equalities of the form $t = t'$, where $t, t' \in \mathbb{T}(\Sigma)$. A closed equality $p = p'$, for some $p, p' \in \mathbb{C}(\Sigma)$, 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 in E.*

In the context of a fixed TSS $\mathcal{T}$, an axiom system E is sound *with respect to a notion of behavioural equivalence $\sim$ if and only if for all $p, p' \in \mathbb{C}(\Sigma)$, if $E \vdash p = p'$, then it holds that $\mathcal{T} \vdash p \sim p'$. It is* ground-complete *if the implication holds in the other direction.*

3 Commutativity Format

Commutativity is an essential property of binary operators specifying that the order of arguments is immaterial. For process algebras, commutativity is defined with respect to a notion of behavioural equivalence. The commutativity format proposed in [31], and given below, guarantees commutativity with respect to any notion of behavioural equivalence that is specified in terms of transitions, e.g., all notions in van Glabbeek's spectrum [21, 20] that include strong bisimilarity.

Definition 5 (Commutativity)**.** *Given a TSS and a binary operator f in its signature, f is called* commutative w.r.t. $\sim$*, if the following equation is sound w.r.t. $\sim$:*

$$f(x_0, x_1) = f(x_1, x_0).$$

Next, we present a syntactic restriction of the tyft format of Groote and Vaandrager [23] that guarantees commutativity w.r.t. any notion of behavioural equivalence that includes strong bisimilarity.

Definition 6 (Comm-form)**.** *A transition system specification over signature Σ is in comm-form format with respect to a set of binary function symbols $COMM \subseteq \Sigma$ if all its f-defining deduction rules with $f \in COMM$ have the following form*

$$\textbf{(d)}\ \frac{\{x_j \xrightarrow{l_{ij}} y_{ij} \mid i \in I\}}{f(x_0, x_1) \xrightarrow{l} t}$$

where $j \in \{0, 1\}$, I is an arbitrary index set, and variables appearing in the source of the conclusion and target of the premises are all pairwise distinct. We denote the set of premises of **(d)** *by H and the conclusion by c. Moreover, for each such rule, there exist a deduction rule* **(d')** *of the following form in the transition system specification*

$$\textbf{(d')}\ \frac{H'}{f(x'_0, x'_1) \xrightarrow{l} t'}$$

and a bijective mapping (substitution) $\hbar$ on variables such that

- $\hbar(x'_0) = x_1$ *and* $\hbar(x'_1) = x_0$,
- $\hbar(t') \sim_{cc} t$ *and*
- $\hbar(h') \in H \cup \{c\}$, *for each* $h' \in H'$,

where $\sim_{cc}$ means equality up to swapping of arguments of operators in $COMM$ in any context. Deduction rule **(d')** *is called the* commutative mirror *of* **(d)**.

To put it informally, the role of substitution $\hbar$ in this definition is to account for the swapping of variables in the source of the conclusion and a possible isomorphic renaming of variables. Thus, the above format requires that when $f \in COMM$, for each f-defining rule, there exists a commutative mirror that enables the "same transitions" when the two arguments of f are swapped. (The format presented in [31] is more general and allows for much more general types of rules and an arbitrary swapping of arguments of n-ary commutative function symbols. We simplified the format to facilitate the presentation here.)

Theorem 7. *If a transition system specification is in* comm-form *format with respect to a set of binary operators COMM, then all operators in COMM are commutative with respect to any notion of behavioural equivalence that includes strong bisimilarity.*

Example 8 (Parallel Composition)**.** *A frequently occurring commutative operator is parallel composition. It appears in, amongst others, ACP [14], CCS [27], and CSP [26, 38]. Here we discuss parallel composition with communication in the style of ACP [14], of which the others are special cases.*

$$\textbf{(p0)}\ \frac{x_0 \xrightarrow{l} y_0}{x_0 \parallel x_1 \xrightarrow{l} y_0 \parallel x_1} \qquad \textbf{(p1)}\ \frac{x_1 \xrightarrow{l} y_1}{x_0 \parallel x_1 \xrightarrow{l} x_0 \parallel y_1}$$

$$\textbf{(p2)}\ \frac{x_0 \xrightarrow{l_0} y_0 \quad x_1 \xrightarrow{l_1} y_1}{x_0 \parallel x_1 \xrightarrow{l} y_0 \parallel y_1} \quad comm(l_0, l_1) = l$$

If the partial synchronization function comm is commutative, then the above TSS is in comm-form *format w.r.t. the singleton set* $COMM = \{\parallel\}$ *and hence it follows from Theorem 7 that* $\parallel$ *is commutative.*

Example 9 (Nondeterministic Choice)**.** *Most process languages for the description of sequential and parallel systems contain a form of nondeterministic choice operator (also called alternative composition). Here we introduce nondeterministic alternative composition as present in CCS [27] and ACP [14].*

$$\textbf{(c0)}\ \frac{x_0 \xrightarrow{l} y_0}{x_0 + x_1 \xrightarrow{l} y_0} \qquad \textbf{(c1)}\ \frac{x_1 \xrightarrow{l} y_1}{x_0 + x_1 \xrightarrow{l} y_1}$$

Using Theorem 7, we can derive that nondeterministic choice is a commutative operator w.r.t. strong bisimilarity or coarser behavioural equivalences. Both of **(c0)** *and* **(c1)** *are in* tyft *format and, taking* $COMM = \{+\}$*, each is the commutative mirror of the other under the mapping* $\hbar(x_0) = x_1$*,* $\hbar(x_1) = x_0$*,* $\hbar(y_1) = y_0$ *and* $\hbar(y_0) = y_1$*.*

4 Associativity Format

Associativity (with respect to a given notion of equivalence), defined below, is an interesting property of binary operators, which does not lend itself easily to syntactic checks like those described in the previous section.

Definition 10 (Associativity)**.** *A binary operator $f \in \Sigma$ is* associative w.r.t. an equivalence $\sim$ *if and only if the following equation is sound w.r.t. $\sim$:*

$$f(x_0, f(x_1, x_2)) = f(f(x_0, x_1), x_2).$$

Because of the nesting of function symbols in associativity axioms, proofs of associativity are usually much more laborious than proofs of both congruence properties and commutativity. For example, proofs of congruence, by and large, make use of induction on the proof structure for transitions and are confined to look at the proof tree *up to depth one*. Thus, they can be performed by looking at deduction rules individually. Proofs of associativity, on the other hand, are usually concerned with proof trees of depth two and hence, if there are n deduction rules for a certain binary operator each with m premises, the number of case distinctions in its associativity proof are n^m in the worst case (for each deduction rule and each premise, there are n possible deduction rules responsible for the transition mentioned in the premise). This is why in [31, Section 5], two of the authors report that their initial attempt to devise a syntactic constraint for associativity did not lead to a concrete rule format. In [18], the same authors (and a co-author) developed an associativity rule format, which is given below.

This format, called ASSOC-de Simone, is a syntactic rule format that guarantees associativity of a binary operator with respect to any notion of behavioural equivalence that is specified in terms of transitions. We take the de Simone format [19] as our starting point and add a number of other ingredients, such as predicates [11], to it. Our choice of the de Simone format is motivated by the inherent complexity of associativity proofs and aims to reduce the size and the number of the proof trees as much as possible. The extensions are motivated by practical examples as illustrated in [18]. Despite the simple setting of our format, the ASSOC-de Simone format is widely applicable to most practical examples we encountered so far. Moreover, we show in [18] that dropping any of the restrictions of the format jeopardizes the meta-result, even for associativity with respect to notions of behavioural equivalence that are coarser than bisimilarity, such as trace equivalence.

Definition 11 (The ASSOC-de Simone Rule Format)**.** *Let $\gamma : L \times L \to L$ be an associative partial function. Consider the following types of rules which are all in the de Simone format.*

1_l. *Left-conforming rules*

$$\frac{x_0 \xrightarrow{l} y_0}{f(x_0, x_1) \xrightarrow{l} f(y_0, x_1)}$$

2_l. *Right-conforming rules*

$$\frac{x_1 \xrightarrow{l} y_1}{f(x_0, x_1) \xrightarrow{l} f(x_0, y_1)}$$

3_l. *Left-choice rules*

$$\frac{x_0 \xrightarrow{l} y_0}{f(x_0, x_1) \xrightarrow{l} y_0}$$

4_l. *Right-choice rules*

$$\frac{x_1 \xrightarrow{l} y_1}{f(x_0, x_1) \xrightarrow{l} y_1}$$

5_l. *Left-choice axioms*

$$\frac{}{f(x_0, x_1) \xrightarrow{l} x_0}$$

6_l. *Right-choice axioms*

$$\frac{}{f(x_0, x_1) \xrightarrow{l} x_1}$$

$7_{(l_0,l_1)}$. *Communicating rules*

$$\frac{x_0 \xrightarrow{l_0} y_0 \quad x_1 \xrightarrow{l_1} y_1}{f(x_0, x_1) \xrightarrow{\gamma(l_0,l_1)} f(y_0, y_1)}$$

A TSS is in the ASSOC-de Simone *format with respect to $f \in \Sigma$ when for each $l \in L$, each f-defining rule is of a type given above and the set of all f-defining rules satisfies the following constraints. (Each proposition P in the following constraints should be read as "there exists a deduction rule of type P in the set of f-defining rules". To avoid repeated uses of parentheses, we assume that $\vee$ and $\wedge$ take precedence over $\Rightarrow$ and $\Leftrightarrow$.)*

1. $5_l \Rightarrow 2_l \wedge 3_l$,
2. $6_l \Rightarrow 1_l \wedge 4_l$,
3. $7_{(l,l')} \Rightarrow \; (1_l \Leftrightarrow 2_{l'}) \wedge (3_l \Leftrightarrow 4_{l'})$
   $\wedge\, (2_l \Leftrightarrow 2_{\gamma(l,l')}) \wedge (4_l \Leftrightarrow 4_{\gamma(l,l')}) \wedge (1_{l'} \Leftrightarrow 1_{\gamma(l,l')}) \wedge (3_{l'} \Leftrightarrow 3_{\gamma(l,l')})$,
4. $1_l \wedge 3_l \Leftrightarrow \exists_{l'}\, \gamma(l, l') = l \wedge 7_{(l,l')} \wedge 5_{l'} \wedge 6_{l'}$,
5. $2_l \wedge 4_l \Leftrightarrow \exists_{l'}\, \gamma(l', l) = l \wedge 7_{(l,l')} \wedge 5_{l'} \wedge 6_{l'}$, *and*
6. $(1_l \vee 4_l) \wedge (2_l \vee 3_l) \Rightarrow (5_l \Leftrightarrow 6_l)$.

The types of rules presented above give us a nice starting point in the development of syntactic conditions on SOS rules that guarantee associativity of operators while covering many practical applications. These rule types cover all rules that are in the de Simone format with four additional restrictions:

1. the target of the conclusion can contain at most one (binary) operator,

2. the aforementioned operator is the same as the one appearing in the source,

3. the labels of the premises and the conclusion coincide (apart from instances of the communicating rule), and

4. testing is disallowed, i.e., if a variable from the source of the conclusion appears in the source of a premise, the target of the same premise must appear in the target of the conclusion.

In [18], we present extensions of these rule types with various syntactic features such as testing and predicates.

Theorem 12. *For a TSS in the* ASSOC-de Simone *format with respect to $f \in \Sigma$, it holds that f is associative for each notion of equivalence $\sim$ that includes strong bisimilarity.*

Next, we present a few applications of the ASSOC-de Simone rule format to operators from the literature.

Example 13 (Parallel composition)**.** *Consider the semantics of ACP parallel composition given in Example 8. Assume that the partial function comm on labels is also associative (as well as commutative). Thus, in terms of the types of deduction rules, we have deduction rules of type 1, 2, and 7. Therefore, the requirements in Definition 11 are met and it can be concluded from Theorem 12 that parallel composition is associative.*

Example 14 (Nondeterministic choice)**.** *Consider the semantics of nondeterministic choice presented in Example 9. In this TSS we find deduction rules of types 3 and 4. Therefore, the requirements of Definition 11 are met and it can be concluded from Theorem 12 that nondeterministic choice is associative.*

Example 15 (Disrupt)**.** *The disrupt operator was originally introduced in the language LOTOS [17], where it is used to model, for example, exception handling. This operator is also used, for instance, in [9], for the description of mode switches.*

$$\frac{x_0 \xrightarrow{l} y_0}{x_0 \blacktriangleright x_1 \xrightarrow{l} y_0 \blacktriangleright x_1} \qquad \frac{x_1 \xrightarrow{l} y_1}{x_0 \blacktriangleright x_1 \xrightarrow{l} y_1}$$

Here we see that only deduction rules of types 1 and 4 are present. As a consequence of Theorem 12 also disrupt is associative.

5 Idempotence Format

Idempotence is a property of binary composition operators requiring that the composition of two identical specifications or programs will result in a piece of specification or program that is equivalent to the original components.

Definition 16 (Idempotence)**.** *A binary operator $f \in \Sigma$ is* idempotent w.r.t. an equivalence $\sim$ *if and only if the following equation is sound w.r.t. $\sim$:*

$$f(x_0, x_0) = x_0.$$

We now present a rule format guaranteeing the idempotence of certain binary operators. It is worth noting that the rule format described below relies on the determinism of certain transition relations. (We recall that a transition relation $\xrightarrow{l}$ is deterministic when, for all closed terms p, p', p'', if $p \xrightarrow{l} p'$ and $p \xrightarrow{l} p''$, then $p' \equiv p''$.) Determinism and idempotence may seem unrelated at first sight. However, it turns out that in order to obtain a powerful rule format for idempotence, we need to have the determinism of certain transition relations in place. Example 20 to follow witnesses the role that determinism plays in applications of our format to operations from the literature.

Definition 17 (The Idempotence Rule Format)**.** *Let $\gamma : L \times L \to L$ be a partial function such that $\gamma(l_0, l_1) \in \{l_0, l_1\}$ if it is defined. We define the following two rule forms.*

1_l. Choice rules

$$\frac{\{x_i \xrightarrow{l} t\} \cup \Phi}{f(x_0, x_1) \xrightarrow{l} t}\ i \in \{0, 1\}$$

$2_{l_0,l_1}$. Communication rules

$$\frac{\{x_0 \xrightarrow{l_0} t_0,\ x_1 \xrightarrow{l_1} t_1\} \cup \Phi}{f(x_0, x_1) \xrightarrow{\gamma(l_0,l_1)} f(t_0, t_1)}\ t_0 \equiv t_1 \text{ or } (l_0 = l_1 \text{ and } \xrightarrow{l_0} \text{ is deterministic })$$

In each case, Φ can be an arbitrary, possibly empty set of formulae.

In addition, we define the starred version of each form, 1^*_l and $2^*_{l_0,l_1}$.

1^*_l. Choice rules

$$\frac{\{x_i \xrightarrow{l} y_i\}}{f(x_0, x_1) \xrightarrow{l} y_i}\ i \in \{0, 1\}$$

$2^*_{l_0,l_1}$**. Communication rules**

$$\frac{\{x_0 \xrightarrow{l_0} y_0,\ x_1 \xrightarrow{l_1} y_1\}}{f(x_0,x_1) \xrightarrow{\gamma(l_0,l_1)} f(y_0,y_1)}\ y_0 \equiv y_1 \text{ or } (l_0 = l_1 \text{ and } \xrightarrow{l_0} \text{ is deterministic })$$

A TSS is in idempotence format w.r.t. a binary operator f *if*

- *each deduction rule is g-defining for some operator g,*
- *each f-defining rule is of the forms 1_l or $2_{l_0,l_1}$, for some $l, l_0, l_1 \in L$, and*
- *for each label $l \in L$ there exists at least one rule of the forms 1^*_l or $2^*_{l,l}$.*

In [3], we give syntactic criteria guaranteeing the determinism of certain transition relations. These syntactic constraints can be used to check the side condition on rules of the form $2_{l_0,l_1}$ and $2^*_{l_0,l_1}$.

Theorem 18. *Assume that a TSS is in the idempotence format with respect to a binary operator f. Then, f is idempotent w.r.t. to any equivalence $\sim$ that includes strong bisimilarity.*

Example 19 (Nondeterministic choice)**.** *Consider again the semantics of nondeterministic choice presented in Example 9. Clearly, the rules given in that example are in the idempotence format w.r.t. +. Hence, it follows from Theorem 18 that + is idempotent w.r.t. any equivalence that includes strong bisimilarity.*

Example 20 (Strong Time-Deterministic Choice)**.** *The choice operator that is used in the timed process algebra ATP [33] has the following deduction rules, where the special label χ denotes the passage of one time unit.*

$$\frac{x_0 \xrightarrow{a} y_0}{x_0 \oplus x_1 \xrightarrow{a} y_0} \qquad \frac{x_1 \xrightarrow{a} y_1}{x_0 \oplus x_1 \xrightarrow{a} y_1} \qquad \frac{x_0 \xrightarrow{\chi} y_0 \quad x_1 \xrightarrow{\chi} y_1}{x_0 \oplus x_1 \xrightarrow{\chi} y_0 \oplus y_1}$$

The idempotence of this operator follows from our format since the last rule for $\oplus$ fits the form $2^*_{\chi,\chi}$ because the transition relation $\xrightarrow{\chi}$ is deterministic over ATP.

6 Deriving Sound and Ground-Complete Axiomatizations

Sound and (ground-)complete axiomatizations are central notions to the algebraic treatment of programming and specification languages and, in particular, to process algebras [14, 26, 27]. 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 the operational semantics modulo bisimilarity). One of the benefits of having complete axiomatizations is that they enable reasoning at the level of syntax without committing to particular semantic models. When the semantic model of behaviour (e.g., the transition system associated to a term) is infinite, these syntactic techniques may come in very handy.

In [4], an automatic method for generating sound and ground-complete axiom systems for strong bisimilarity over the transition systems induced by GSOS language specifications is presented. It is assumed that there are only transition systems $\longrightarrow \subseteq \mathbb{C}(\Sigma) \times L \times \mathbb{C}(\Sigma)$. In the remainder, we give a short overview of the technique presented in [4] and point out several variants and extensions thereof in the literature.

The approach of [4] is based on a restricted form of SOS deduction rules, called the GSOS format [16]. In what follows, for the sake of simplicity and uniformity, we only consider the positive subset of the GSOS format, defined below.

Definition 21. *(Positive GSOS Format) A deduction rule is in the* positive GSOS format *when it is of the following form.*

$$\frac{\{x_i \xrightarrow{a_{ij}} y_{ij} \mid i \in I, j \in J_i\}}{f(\vec{x}) \xrightarrow{a} t}$$

where

- *the variables in* $\vec{x} = (x_0, \ldots, x_{n-1})$ *and* $\{y_{ij} \mid i \in I, j \in J_i\}$ *are all pairwise distinct,*
- I *is a subset of* $\{0, \ldots, n-1\}$*, where* n *is the arity of* f*,*
- J_i *is a finite index set, for each* $i \in I$*, and*
- $vars(t) \subseteq \{x_0, \ldots, x_{n-1}\} \cup \{y_{ij} \mid i \in I, j \in J_i\}$.

It is well-known that some operators whose operational semantics can be expressed in the positive GSOS format cannot be finitely axiomatized modulo bisimilarity. Therefore, in order to axiomatize them finitely, one needs auxiliary operators. (See, e.g., [29].) Thus, in order to achieve a finite ground-complete axiomatization, we may need to extend the signature of a language with fresh operators. The kind of extension required for this purpose is called disjoint extension and is defined below.

Definition 22 (Disjoint Extension)**.** *Consider TSSs* $\mathcal{T}_1 = (\Sigma_1, L_1, D_1)$ *and* $\mathcal{T}_2 = (\Sigma_2, L_2, D_2)$*. TSS* $\mathcal{T}_2$ *is a* disjoint extension *of TSS* $\mathcal{T}_1$ *if and only if* $\Sigma_1 \subseteq \Sigma_2$*,* $D_1 \subseteq D_2$ *and the operators from* Σ_1 *do not occur in the sources of the deduction rules from* $D_2 \setminus D_1$*.*

The crucial property of the notion of disjoint extension that underlies the developments to follow is that if $\mathcal{T}_2$ is a disjoint extension of $\mathcal{T}_1$, then two closed terms over Σ_1 are strongly bisimilar w.r.t. $\mathcal{T}_1$ if and only if they are strongly bisimilar w.r.t. $\mathcal{T}_2$. This means that an axiom system that is sound and ground-complete w.r.t. bisimilarity over $\mathcal{T}_2$ can be used to show all the valid equalities between closed terms over the signature of $\mathcal{T}_1$.

Next, we define when a TSS does not allow infinite traces. Such a TSS is called *trace finite*. In [4], syntactic criteria are given for guaranteeing trace finiteness of a TSS.

Definition 23 (Trace Finiteness)**.** *Let* $\mathcal{T}$ *be a TSS in the positive* GSOS *format. A term* $p \in \mathbb{C}(\Sigma)$ *is* trace finite *iff there exists no infinite sequence* $p_0, l_0, p_1, l_1, \ldots$ *of closed terms* p_i *and labels* l_i *such that* $p \equiv p_0$ *and* $p_i \xrightarrow{l_i} p_{i+1}$*, for all* $i \geq 0$*. The TSS* $\mathcal{T}$ *is* trace finite *iff all terms in* $\mathbb{C}(\Sigma)$ *are trace finite.*

The following theorem from [4] states that for trace-finite TSSs in the positive GSOS format, we can always add sufficiently many auxiliary operators in order to generate a finite ground-complete axiomatization of strong bisimilarity. The procedure to obtain such a ground-complete axiomatization is sketched subsequently.

Theorem 24. *Let* $\mathcal{T}$ *be a trace-finite TSS in the* GSOS *format. Then, there are a disjoint extension* $\mathcal{T}'$ *of* $\mathcal{T}$ *and a finite axiom system* E' *such that* E' *is a sound and ground-complete axiomatization of bisimilarity on closed terms from* $\mathcal{T}'$*.*

The approach of [4] relies on the presence of three basic operators in the signature, namely a constant $\mathbf{0}$, denoting inaction or deadlock, a unary action prefixing operator $a.\_$, for each $a \in L$, and a binary nondeterministic choice $\_ + \_$, already presented in Example 9. These operators allow one to denote all finite synchronization trees in the sense of Milner [27].

The inaction constant $\mathbf{0}$ has no transition and hence it has no defining rule in the semantics. The deduction rule for action prefixing is given below and the deduction rules for nondeterministic choice are those presented in Example 9.

$$\frac{}{a.x_0 \xrightarrow{a} x_0}$$

These basic operators are finitely axiomatized modulo bisimilarity by the following axiom system [25].

$$x_0 + x_1 = x_1 + x_0$$
$$(x_0 + x_1) + x_2 = x_0 + (x_1 + x_2)$$
$$x_0 + x_0 = x_0$$
$$x_0 + \mathbf{0} = x_0$$

Note that the above axioms are sound not only for the small TSS containing only the three operators introduced above, but also for any disjoint extension thereof. Hence, if the TSS to be axiomatized does not contain these operators, we can safely add them to the signature and the above-given axiom system remains sound in this extended TSS as well as in all of its disjoint extensions. All the equations that are generated by the method in [4] are "robust", in the sense that they remain valid for any disjoint extension of a GSOS language.

The idea behind the procedure for the automatic generation of finite axiomatizations of bisimilarity presented in [4] is as follows. Assume that we have a TSS $\mathcal{T}$ in (positive) GSOS format that disjointly extends the TSS described above for finite synchronization trees. Since we already have the above-given complete axiomatization of bisimilarity over finite synchronization trees, in order to obtain a ground-complete axiomatization of bisimilarity over $\mathcal{T}$, it suffices only to generate a disjoint extension of $\mathcal{T}$ and a finite axiom system that can be used to rewrite each closed term into an equivalent finite synchronization tree.

The axiomatization procedure starts with the axiom system given above. Then, on top of these laws, for each operator we generate new axioms by using the above-mentioned operators and other auxiliary ones. Some operators, namely the smooth and distinctive ones introduced in Definition 26 to follow, distribute w.r.t. nondeterministic choice in some of their arguments and can be handled without recourse to auxiliary operators. Other operators, namely the smooth ones that are not distinctive, can be expressed as nondeterministic compositions of auxiliary operators that are smooth and distinctive. Others still are expressed in terms of auxiliary smooth operators with possibly different arity.

Definition 25. *(Smooth Operators) An n-ary function symbol f is called* smooth *when all f-defining deduction rules are of the following form*

$$\frac{\{x_i \xrightarrow{a_i} y_i \mid i \in I\}}{f(x_0, \ldots, x_{n-1}) \xrightarrow{a} C[\overrightarrow{x}, \overrightarrow{y}]},$$

where $I \subseteq \{0, .., n-1\}$, $C[\overrightarrow{x}, \overrightarrow{y}]$ is a term containing only variables in $\overrightarrow{x}$ and $\overrightarrow{y}$ and moreover no x_i appearing in the source of a premise appears in the target of the conclusion.

Definition 26. *(Distinctive Operators) A smooth operator is called* distinctive*, when for each two distinct f-defining rules of the following form*

$$\frac{\{x_i \xrightarrow{a_i} y_i \mid i \in I\}}{f(x_0, \ldots, x_{n-1}) \xrightarrow{a} C[\overrightarrow{x}, \overrightarrow{y}]} \qquad \frac{\{x'_i \xrightarrow{a'_i} y'_i \mid i \in I'\}}{f(x'_0, \ldots, x'_{n-1}) \xrightarrow{a'} C'[\overrightarrow{x'}, \overrightarrow{y'}]},$$

it holds that $I = I'$ and there exists an $i \in I$ such that $a_i \neq a'_i$.

Example 27. *The parallel composition operator of Example 8 is smooth but not distinctive.*

To axiomatize a distinctive operator f, it suffices to add the following equations, which describe the interplay between f and the operations for finite synchronization trees, to our axiom system.

1. Distributivity laws: If $i \in I$ for each deduction rule of the form given in Definition 25, we have the equation:

   $$f(x_0, \ldots, x_i + x'_i, \ldots, x_{n-1}) = f(x_0, \ldots, x_i, \ldots, x_{n-1}) + f(x_0, \ldots, x'_i, \ldots, x_{n-1}).$$

2. Action laws: For each f-defining rule of the form given in Definition 25, we have the equation

   $$f(P_0, \ldots, P_{n-1}) = a.C[\overrightarrow{P}, \overrightarrow{y}],$$

   where $P_i = a_i.y_i$, if $i \in I$, and $P_i = x_i$, otherwise.

3. Inaction laws: We have a law of the form

   $$f(P_0, \ldots, P_{n-1}) = \mathbf{0},$$

   where

   - each P_i is of the form $\mathbf{0}$, x_i or $b_i.x_i$ for some label b_i and
   - for each deduction rule of the form given in Definition 25, there is an index $i \in I$ such that $P_i = \mathbf{0}$ or $P_i = b_i.x_i$, for some $b_i \neq a_i$.

The axiom system obtained thus far gives a sound and ground-complete axiomatization of smooth and distinctive operators (possibly over a disjoint extension of the original language with inaction, action prefixing and nondeterministic choice). In order to axiomatize smooth operators that are not distinctive, we partition their deduction rules into sets of rules satisfying the criteria of Definition 26. This is always possible and in the worst case singleton sets of rules trivially

satisfy these criteria. Then, for each partition, we introduce an auxiliary operator with the defining rules given in that partition. The introduced operators are by construction smooth and distinctive. Hence, we can apply the construction given before to these operators to axiomatize them.

Assume that for a smooth but non-distinctive operator f, the set of deduction rules is partitioned into n sets and thus n smooth and distinctive function symbols f_0 to f_{n-1} are introduced. Then, in addition to the axiom system for each auxiliary operator, we introduce the following axiom.

$$f(x_0, \ldots, x_{n-1}) = f_0(x_0, \ldots, x_{n-1}) + \cdots + f_{n-1}(x_0, \ldots, x_{n-1})$$

The above axiom together with the axioms generated for $f_0, \ldots, f_{n-1}$ give a sound and ground-complete axiomatization for the smooth operator f.

The following example illustrates this procedure.

Example 28. *Consider again the parallel composition operator of Example 8. The trivial partitioning of its deduction rules gives rise to 3 auxiliary operators* $\|_0$, $\|_1$, *and* $\|_2$, *with the following semantics.*

$$\textbf{(p0)}\ \frac{x_0 \xrightarrow{l} y_0}{x_0 \,\|_0\, x_1 \xrightarrow{l} y_0 \,\|\, x_1} \qquad \textbf{(p1)}\ \frac{x_1 \xrightarrow{l} y_1}{x_0 \,\|_1\, x_1 \xrightarrow{l} x_0 \,\|\, y_1}$$

$$\textbf{(p2)}\ \frac{x_0 \xrightarrow{l_0} y_0 \quad x_1 \xrightarrow{l_1} y_1}{x_0 \,\|_2\, x_1 \xrightarrow{l} y_0 \,\|\, y_1} \quad comm(l_0, l_1) = l$$

Then, applying the procedure given before, we obtain the following axiom system.

$$\begin{array}{llll}
x_0 + x_1 = x_1 + x_0 & & (x_0 + x_1) + x_2 = x_0 + (x_1 + x_2) & \\
x_0 + x_0 = x_0 & & x_0 + \mathbf{0} = x_0 & \\
(a.x_0) \,\|_0\, x_1 = a.(x_0 \,\|\, x_1) & & x_0 \,\|_1 (a.x_1) = a.(x_0 \,\|\, x_1) & \\
(a.x_0) \,\|_2 (b.x_1) = c.(x_0 \,\|\, x_1) & \text{if } comm(a,b) = c & & \\
\mathbf{0} \,\|_0\, x_1 = \mathbf{0} & & x_0 \,\|_1\, \mathbf{0} = \mathbf{0} & \\
\mathbf{0} \,\|_2\, x_1 = \mathbf{0} & & x_0 \,\|_2\, \mathbf{0} = \mathbf{0} & \\
(a.x_0) \,\|_2 (b.x_1) = \mathbf{0} & \text{if } comm(a,b) \text{ is undefined} & &
\end{array}$$

$$\begin{array}{c}
(x_0 + x_0') \,\|_0\, x_1 = (x_0 \,\|_0\, x_1) + (x_0' \,\|_0\, x_1) \\
x_0 \,\|_1 (x_1 + x_1') = (x_0 \,\|_1\, x_1) + (x_0 \,\|_1\, x_1') \\
(x_0 + x_0') \,\|_2\, x_1 = (x_0 \,\|_2\, x_1) + (x_0' \,\|_0\, x_1) \\
x_0 \,\|_2 (x_1 + x_1') = (x_0 \,\|_2\, x_1) + (x_0 \,\|_2\, x_1') \\
x_0 \,\|\, x_1 = x_0 \,\|_0\, x_1 + x_0 \,\|_1\, x_1 + x_0 \,\|_2\, x_1
\end{array}$$

The generated axioms do resemble the original axioms of [14] to a large extent. The auxiliary operators $\|_0$, $\|_1$ *and* $\|_2$ *are called left, right and communication merge in the literature.*

Using the techniques introduced so far, one can axiomatize smooth operators. Non-smooth operators may test some of their arguments or make copies of them. As described in detail in [4, Section 4], we can axiomatize an n-ary non-smooth operator f by means of an m-ary smooth operator g that simulates all the copying and testing done by the rules for f. The following example illustrates this, rather technical, procedure on a simple non-smooth operation.

Example 29. *Consider a (hypothetical) unary operator f with the following operational semantics.*

$$\frac{x_0 \xrightarrow{a} y_0 \quad x_0 \xrightarrow{b} y_0'}{f(x_0) \xrightarrow{a} g(x_0, y_0')}$$

The function symbol f is not smooth, because x_0 appears twice as the source of premises and, moreover, both x_0 and the result of one of its transitions, i.e., y_0', appear in the target. To remedy this, we introduce a binary auxiliary operator h with the following semantics.

$$\frac{x_0 \xrightarrow{a} y_0 \quad x_1 \xrightarrow{b} y_1}{h(x_0, x_1) \xrightarrow{a} g(x_0, y_1)}$$

The function symbol h is now smooth and thus, the procedure given before can readily axiomatize it. Adding the following equation will complete the axiomatization of f:

$$f(x_0) = h(x_0, x_0).$$

To conclude, using the procedure sketched above, by adding sufficiently many auxiliary operators, one can finitely axiomatize bisimilarity over closed terms in any TSS in the positive GSOS format.

A generalization of Theorem 24 to non-trace-finite and non-positive TSSs is also presented in [4]; the generalization to non-trace-finite TSSs requires the addition of an infinitary conditional equation, the *Approximation Induction Principle* from [10], to the generated axiom system.

The techniques from [4] were extended in [13] 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 [4]. The resulting format is called the TAGH format. (The acronym TAGH format stands for *termination and GSOS hybrid* format.)

The definition of languages in the GSOS and the TAGH formats requires that the signature, the set of action labels, and the set of deduction rules be finite. In [1], 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. The sub-format regular GSOS guarantees that every closed term describes a finite labelled transition system [2]. For this sub-format, Aceto presents a variation on the procedure described above that allows one to generate a sound and ground-complete axiomatization for bisimilarity, which makes use of the *Recursive Specification Principle* [12].

Bloom [15] has shown that the approach of [4] can also be used for generating axioms for rooted branching bisimilarity, and van Glabbeek claims in [22] that this is also the case for rooted-η bisimilarity. The latter work also offers an adaptation of the approach of [4] that yields finite, sound and ground-complete axiomatizations for rooted branching and rooted delay bisimilarities [20].

Axiom systems for preorders have been generated, too, see for instance [39]. Along the same lines, [40] generates prioritized rewrite systems for TSSs with an ordering on deduction rules (see [30]).

7 Conclusions and Open Problems

In this paper, we have presented an overview of some of the existing meta-results for guaranteeing the validity of algebraic properties and for generating ground-complete axiom systems from operational semantics. There is an ongoing research in this field and we currently have proposals for rule formats for unit and zero elements; see [8]. A closely related line of research aims at developing (possibly automatic) proof techniques for establishing the soundness of axiom systems using the SOS specification of a language; see, e.g., the papers [19, 42, 5].

There remain many open problems to be addressed. An extension of the ASSOC-de Simone format with negative premises seems a challenging research problem to us. Also, relaxing the formats of Sections 3-5 to guarantee algebraic properties for weaker notions of behavioural equivalence than bisimilarity could be worth investigating. If a TSS conforms to any of the rule formats presented in this paper, any disjoint extension of such a TSS also conforms to the same format. Hence, the algebraic properties proven by the meta-theorems are robust, in the sense that they remain sound under any disjoint extension. It is in general very challenging and interesting to address the robustness of algebraic properties under extensions of TSSs. Another interesting topic for future research is to combine the techniques of Sections 3-5 with those presented in Section 6 in order to generate "more natural" axiomatizations, which resemble the axiomatizations presented so far by the language designers. A challenging open problem in this research area is the development of methods for the automatic generation of axiomatizations of behavioural equivalences that are complete over arbitrary open terms. The hardness of this problem is witnessed by the lack of such results even for specific process algebras that contain operators like restriction and parallel composition

with synchronization. See [6] for a survey of results on complete axiomatizations up to 2005.

References

[1] Luca Aceto. Deriving complete inference systems for a class of GSOS languages generating regular behaviours. In Bengt Jonsson and Joachim Parrow, editors, *Proceedings of the fifth International Conference on Concurrency Theory (CONCUR'94)*, volume 836 of *Lecture Notes in Computer Science*, pages 449–464. Springer-Verlag, Berlin, Germany, 1994.

[2] Luca Aceto. GSOS and finite labelled transition systems. *Theoretical Computer Science*, 131(1):181–195, 1994.

[3] Luca Aceto, Arnar Birgisson, Anna Ingolfsdottir, MohammadReza Mousavi, and Michel A. Reniers. Rule formats for determinism and idempotence. In *Proceedings of the 3rd International Conference on Fundamentals of Software Engineering (FSEN'09)*, Lecture Notes in Computer Science, Kish Island, Iran, 2009. Springer-Verlag, Berlin, Germany.

[4] Luca Aceto, Bard Bloom, and Frits W. Vaandrager. Turning SOS rules into equations. *Information and Computation (I&C)*, 111:1–52, 1994.

[5] Luca Aceto, Matteo Cimini, and Anna Ingolfsdottir. A bisimulation-based method for proving the validity of equations in GSOS languages. In *Proceedings of the Workshop on Structural Operational Semantics (SOS'09)*, Electronic Proceedings in Theoretical Computer Science, 2009.

[6] Luca Aceto, Wan Fokkink, Anna Ingolfsdottir, and Bas Luttik. Finite equational bases in process algebra: Results and open questions. In Aart Middeldorp, Vincent van Oostrom, Femke van Raamsdonk, and Roel C. de Vrijer, editors, *Processes, Terms and Cycles: Steps on the Road to Infinity, Essays Dedicated to Jan Willem Klop, on the Occasion of His 60th Birthday*, volume 3838 of *Lecture Notes in Computer Science*, pages 338–367. Springer-Verlag, 2005.

[7] Luca Aceto, Willem Jan (Wan) Fokkink, and Chris Verhoef. Structural operational semantics. In Jan A. Bergstra, Alban Ponse, and Scott A. Smolka, editors, *Handbook of Process Algebra, Chapter 3*, pages 197–292. Elsevier Science, Dordrecht, The Netherlands, 2001.

[8] Luca Aceto, Anna Ingolfsdottir, MohammadReza Mousavi, and Michel A. Reniers. A rule format for unit elements. In *Proceedings of the 36th International Conference on Current Trends in Theory and Practice of Computing (SOFSEM 2010)*, Lecture Notes in Computer Science, Springer-Verlag, 2010.

[9] J. C. M. Baeten and J. A. Bergstra. Mode transfer in process algebra. Technical Report CS-R 00-01, Department of Computer Science, Eindhoven University of Technology, 2000.

[10] J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. Verification of an alternating bit protocol by means of process algebra. In *Proceedings Mathematical Models of Specification and Synthesis of Software Systems 1985*, volume 215 of *Lecture Notes in Computer Science*, pages 9–23. Springer-Verlag, 1986.

[11] J.C.M. (Jos) Baeten and Chris Verhoef. A congruence theorem for structured operational semantics with predicates. In Eike Best, editor, *International Conference on Concurrency Theory (CONCUR'93)*, volume 715 of *Lecture Notes in Computer Science*, pages 477–492. Springer-Verlag, Berlin, Germany, 1993.

[12] J.C.M. (Jos) Baeten and W. Peter Weijland. *Process Algebra*, volume 18 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1990.

[13] Jos C.M. Baeten and Erik P. de Vink. Axiomatizing GSOS with termination. *Journal of Logic and Algebraic Programming (JLAP)*, 60-61:323–351, 2004.

[14] Jan A. Bergstra and Jan Willem Klop. Process algebra for synchronous communication. *Information and Control*, 60(1-3):109–137, 1984.

[15] Bard Bloom. Structural operational semantics for weak bisimulations. *Theoretical Computer Science (TCS)*, 146:25–68, 1995.

[16] Bard Bloom, Sorin Istrail, and Albert R. Meyer. Bisimulation can't be traced. *Journal of the ACM (JACM)*, 42(1):232–268, January 1995.

[17] Ed Brinksma. A tutorial on LOTOS. In Michel Diaz, editor, *Proceedings of Protocol Specification, Testing and Verification V*, pages 171Ű–194. North-Holland, 1985.

[18] Sjoerd Cranen, MohammadReza Mousavi, and Michel A. Reniers. A rule format for associativity. In Franck van Breugel and Marsha Chechik, editors, *Proceedings of the 19th International Conference on Concurrency Theory (CONCUR'08)*, volume 5201 of *Lecture Notes in Computer Science*, pages 447–461, Toronto,Canada, 2008. Springer-Verlag, Berlin, Germany.

[19] Robert de Simone. Higher-level synchronizing devices in MEIJE-SCCS. *Theoretical Computer Science (TCS)*, 37:245–267, 1985.

[20] Robert Jan (Rob) van Glabbeek. The linear time - branching time spectrum II. In Eike Best, editor, *International Conference on Concurrency Theory (CONCUR'93)*, volume 715 of *Lecture Notes in Computer Science*, pages 66–81. Springer-Verlag, Berlin, Germany, 1993.

[21] Robert Jan (Rob) van Glabbeek. The linear time - branching time spectrum I. In Jan A. Bergstra, Alban Ponse, and Scott A. Smolka, editors, *Handbook of Process Algebra, Chapter 1*, pages 3–100. Elsevier Science, Dordrecht, The Netherlands, 2001.

[22] Robert Jan (Rob) van Glabbeek. On cool congruence formats for weak bisimulations (extended abstract). In *Proceedings of the 2nd International Colloquium on Theoretical Aspects of Computing (ICTAC'05)*, volume 3722 of *Lecture Notes in Computer Science*, pages 318–333. Springer-Verlag, Berlin, Germany, 2005.

[23] Jan Friso Groote and Frits W. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Information and Computation (I&C)*, 100(2):202–260, October 1992.

[24] Matthew Hennessy and Gordon D. Plotkin. Full abstraction for a simple parallel programming language. In Jirí Becvár, editor, *Proceedings of the 8th Symposium on Mathematical Foundations of Computer Science (MFCS'79)*, volume 74 of *Lecture Notes in Computer Science*, pages 108–120. Springer-Verlag, Berlin, Germany, 1979.

[25] Matthew Hennessy and A.J.R.G. (Robin) Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32(1):137–161, 1985.

[26] C.A.R. (Tony) Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[27] A.J.R.G. (Robin) Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, 1980.

[28] A.J.R.G. (Robin) Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[29] Faron Moller. The nonexistence of finite axiomatisations for CCS congruences. In *Proceedings, Fifth Annual IEEE Symposium on Logic in Computer Science, 4–7 June 1990, Philadelphia, Pennsylvania, USA*, pages 142–153. IEEE Computer Society, 1990.

[30] MohammadReza Mousavi, Iain C.C. Phillips, Michel Reniers, and Irek Ulidowski. Semantics and expressiveness of ordered SOS. *Information and Computation (I&C)*, 207(2):85–119, 2009.

[31] MohammadReza Mousavi, Michel Reniers, and Jan Friso Groote. A syntactic commutativity format for SOS. *Information Processing Letters (IPL)*, 93:217–223, March 2005.

[32] MohammadReza Mousavi, Michel A. Reniers, and Jan Friso Groote. SOS formats and meta-theory: 20 years after. *Theoretical Computer Science*, 3(373):238–272, 2007.

[33] Xavier Nicollin and Joseph Sifakis. The algebra of timed processes ATP: theory and application. *Information and Computation (I&C)*, 114(1):131–178, October 1994.

[34] David M.R. Park. Concurrency and automata on infinite sequences. In *Proceedings of the 5th GI Conference*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer-Verlag, Berlin, Germany, 1981.

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

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

[37] Gordon D. Plotkin. A structural approach to operational semantics. *Journal of Logic and Algebraic Programming (JLAP)*, 60:17–139, 2004. This article first appeared as [35].

[38] Bill Roscoe. *The Theory and Practice of Concurrency*. Prentice Hall, 1997.

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

[40] Irek Ulidowski. Rewrite systems for OSOS process languages. In *Proceedings of the 14th International Conference on Concurrency Theory (CONCUR'03)*, volume 2761 of *Lecture Notes in Computer Science*, pages 87–102. Springer-Verlag, 2003.

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

[42] Muck van Weerdenburg. Automating soundness proofs. In *Proceedings of the Workshop on Structural Operational Semantics (SOS'08)*, volume 229 of *Electronic Notes in Theoretical Computer Science*, pages 107–118, 2009.