SOS Meta-theory: A Very Short Introduction

Luca Aceto
School of Computer Science
Reykjavik University
Kringlan 1, 103 Reykjavik, Iceland

In all fields of science, researchers make progress by first studying in detail several concrete scenarios and problems. Once these specific examples have been well understood, the next step is often to isolate the characteristics that they share and the common techniques that underlie their study. This brings about a key step in any form of scientific endeavour, namely that of *abstraction*. We use abstraction to get to the core of things, to strip the concrete scenarios of the details that are immaterial for certain properties to hold and results to apply, to realize that some specific phenomena may be found in several different settings simply because, albeit possibly very different, those settings have some basic principles in common.

In mathematics, one possible, and very classic, approach to generalizing results is provided by the *axiomatic method*, as pioneered by Euclid already over 2000 years ago. Once proved from Euclid's axioms for Euclidean geometry, a theorem holds true in *every structure* that satisfies the axioms. As Hilbert famously stated, it does not really matter whether the symbols in the axioms are interpreted as points and lines, or as tables and chairs. The only thing that matters is that we have a *model* of the axioms.

Now you may ask, 'What does this have to do with our topic? Can this approach to abstraction, namely the axiomatic method, be used to study the languages used in computer science and, in particular, process description languages like CCS, CSP and so on?'

In some of your courses and/or in your research, you have seen a wealth of results applying to specific programming or specification languages. For instance, the textbook [2] offers several results for the language CCS [8]. An example of such a result is Theorem 3.2 in [2] to the effect that strong bisimilarity is a

congruence over CCS. Moreover, those amongst you who solved Exercise 3.15 in [2] have already seen that strong bisimilarity is also preserved by at least one other operator that is not included in the syntax for the language CCS. But there are many more operators out there for which this property holds! You should, for instance, be able to convince yourself with very little effort that the (not-so-useful) operation a-and-b( ) defined by the rule:

$$\frac{P \stackrel{a}{\rightarrow} P_1,\ P \stackrel{b}{\rightarrow} P_2}{\text{a-and-b}(P) \stackrel{a}{\rightarrow} P_1 + P_2}$$

preserves bisimilarity—that is, that a-and-b$(P) \sim$ a-and-b(Q) holds whenever $P \sim Q$. (Do so!)

Can we generalize these congruence results? What do we need to do so? And how do we even approach such a question? Let us start by defining a general framework that can be used to abstract the essence of the concrete languages you have met so far.

We begin by observing that all languages consist of a *syntax* and a *semantics*. The syntax of a language can be described by giving the collection of operations it offers for constructing new 'programs' from ones that we have already built. For example, the syntax of CCS can be specified by listing the collection of operations used in the BNF grammar for that language, together with the number of arguments that each such operation is supposed to take. Following the terminology of (universal) algebra, we call this set of operations the *signature* for the language CCS.

In general, a signature Σ consists of a collection of operation symbols together with their *arities*—that is, the number of arguments the operation symbol takes. An operation symbol of arity zero is usually called a *constant* symbol. For instance, **0** is a constant symbol in the signature for CCS.

Example 1 The signature Σ_{Nat} consists of the constant Z, the unary operation symbol succ and the binary operation symbol $\vee$. In what follows, we will write $\vee$ in infix form—that is, we use $t \vee u$ instead of $\vee(t, u)$. ♦

Example 2 The signature Σ_{BPA} consists of the constants δ, ε and a, and the binary operation symbols $\cdot$ and $+$. In what follows, we will write $t \cdot u$ and $t + u$ instead of $\cdot(t, u)$ and $+(t, u)$, respectively.

We will sometimes refer to the language with signature Σ_{BPA} as BPA (for Basic Process Algebra)—see, e.g., [4]. ♦

In general, given a signature Σ and a collection of variables V, the language associated with Σ is the set of terms $T(\Sigma)$ that can be constructed using the following rules:

- if $x \in V$ then $x \in T(\Sigma)$,
- if $n \geq 0$, $t_1, \ldots, t_n \in T(\Sigma)$ and $f \in \Sigma$ has arity n, then $f(t_1, \ldots, t_n) \in T(\Sigma)$ and
- nothing else is in $T(\Sigma)$.

The set of terms $T(\Sigma)$ can be equivalently specified by the BNF grammar below:

$$t ::= x \mid f(t_1, \ldots, t_n) \ ,$$

where $x \in V$, and $f \in \Sigma$ has arity n.

We say that a term $t \in T(\Sigma)$ is *closed*, when it does not contain occurrences of variables.

Exercise 1 *Give examples of closed terms in $T(\Sigma_{Nat})$ and $T(\Sigma_{BPA})$. How many closed terms does $T(\Sigma)$ contain if the signature Σ has no constant symbols?* ♦

A (closed) substitution maps variables in V to (closed) terms. For every term t and substitution σ, the term $\sigma(t)$ is obtained by replacing every occurrence of a variable x in t by $\sigma(x)$. Note that $\sigma(t)$ is closed if σ is a closed substitution.

We now turn to a general tool for the description of the semantics of a language. Our aim is to generalize the rule-based approach that we have followed in giving the semantics of a calculus like CCS. In this approach, the semantics of a language is given by a labelled transition system (see Definition 2.1 on page 19 in [2]) whose states are the closed terms in $T(\Sigma)$, and whose transitions between terms are those that can proved to exist using a collection of rules R.

By way of example, let us consider the collection of rules on Table 1, which describe the operational semantics of the operations in the signature Σ_{BPA}. Intuitively, transitions of the form $P \xrightarrow{\checkmark} P'$ indicate that process P may terminate its computation successfully, whereas transitions of the form $P \xrightarrow{a} P'$ say that process P may perform action a thereby evolving into process P'. So, for instance, process a can perform action a and then terminate successfully since it affords the following sequence of transitions:

$$a \xrightarrow{a} \varepsilon \xrightarrow{\checkmark} \delta \ .$$

$$\text{ACT}\ \frac{}{a \xrightarrow{a} \varepsilon}$$

$$\text{TERM}\ \frac{}{\varepsilon \xrightarrow{\checkmark} \delta}$$

$$\text{SUM1}\ \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}$$

$$\text{SUM2}\ \frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$$

$$\text{SEQ1}\ \frac{P \xrightarrow{a} P'}{P \cdot Q \xrightarrow{a} P' \cdot Q}$$

$$\text{SEQ2}\ \frac{P \xrightarrow{\checkmark} P' \quad Q \xrightarrow{\alpha} Q'}{P \cdot Q \xrightarrow{\alpha} Q'}$$

Table 1: SOS rules for $T(\Sigma_{\text{BPA}})$ ($\alpha \in \{a, \checkmark\}$)

Exercise 2 *Prove that strong bisimilarity is preserved by all the operators in Σ_{BPA}—that is, show that if $P_1 \sim Q_1$ and $P_2 \sim Q_2$ then $P_1 + P_2 \sim Q_1 + Q_2$ and $P_1 \cdot P_2 \sim Q_1 \cdot Q_2$ both hold.* ⧫

The rules on Table 1, like those on Table 2.2 on page 24 in [2] have the general form below.

$$\frac{\text{Premises}}{\text{Conclusion}}$$

In the above rule template, we expect that both the premises and the conclusion will describe constraints on the transitions that are possible for closed terms. However, it is somewhat unlikely that we can develop a useful theory of semantic language specifications by considering rule templates of such generality. How can we

achieve a reasonable level of abstraction while at the same time precisely defining what rules are allowed, so that we can prove theorems about classes of languages using them? Once more, we draw inspiration from the classic axiomatic method and logic. We are going to define a collection of constraints over rules, and then show that any operation or language, whose semantics is given by rules that obey those constraints, has some desirable property that we intend to investigate. This general approach is best introduced by means of a concrete example.

Let us look once more at the rules describing the operational semantics of the operations of CCS and BPA. All those rules define the transitions of a composite term in terms of the transitions that are possible for their immediate components. For instance, the initial transitions of a term of the form $P \cdot Q$ can be inferred using rules SEQ1–2 on Table 1 from the initial transitions of P and Q. Moreover, in the conclusion of a rule, the term that is the target of the transition does not contain any meta-variable that is the source of a premise. For example, in rule SEQ1 on Table 1, the target of the conclusion of the rule, namely $P' \cdot Q$, does not contain P. Note, however, that $P' \cdot Q$ contains P', which is a meta-variable standing for the target of the transition of P that is the premise of rule SEQ1. Intuitively, terms whose transitions are premises of rules are 'consumed' during that transition and are not used in the target of the conclusions of operational rules. Another feature of the rules we have seen so far is that the meta-variables in a rule are all pairwise distinct. Finally, we note that in each of the rules for CCS and BPA, there is at most one premise describing assumptions on the behaviour of each of the arguments of an operation, and each meta-variable occurs in the target of a rule at most once.

The analysis above suggests that we look at rules having the following form:

$$\frac{\{x_i \xrightarrow{a_i} y_i \mid i \in I\}}{f(x_1, \ldots, x_n) \xrightarrow{a} t} \ , \tag{1}$$

where

- I is included in $\{1, \ldots, n\}$,
- a and a_i, $i \in I$, are actions,
- the variables $x_1, \ldots, x_n$ and y_i, $i \in I$, are all pairwise distinct and
- t is a term over our signature of interest which may contain variables y_i, $i \in I$, and those x_j with $j \in \{1, \ldots, n\} - I$. Moreover, t contains each variable at most once.

You should pause for a moment, and check that all of the rules for CCS and BPA satisfy the requirements above. Note, moreover, that if f is a constant symbol, then each rule of the form (1) for f is an axiom—that is, it has no premises.

The constraints on operational rules laid out in (1) are an example of a *rule format*. A rule format gives criteria on operational rules that we expect those rules to fulfill. The rule format described by rules like those in (1) is usually referred to as the *de Simone format* because it was introduced by de Simone in his seminal paper [10]. To the best of my knowledge, it is the first example ever of a rule format in the field of Structural Operational Semantics.

Definition 1 A *de Simone language* is a pair (Σ, R), where Σ is a signature and R is a collection of rules over Σ of the form (1). ⧫

As mentioned before, a de Simone language (Σ, R) is meant to describe a labelled transition system whose states are the closed terms that can be constructed using the operations in the signature Σ, and whose transitions between closed terms are those that can be proved using the rules in R. For the sake of clarity, we now proceed to define precisely the collection of transitions that can be proved from R.

Definition 2 Let (Σ, R) be a de Simone language. The collection of transitions provable from R is the smallest set T of transitions of the form $P \xrightarrow{a} P'$, where P and P' are closed terms over Σ and a is an action, that satisfies the following property:

> If there are a rule in R of the form (1), and a closed substitution σ such that $\sigma(x_i) \xrightarrow{a_i} \sigma(y_i)$ is in T for each $i \in I$, then $\sigma(f(x_1, \ldots, x_n)) \xrightarrow{a} \sigma(t)$ is also in T.

⧫

Exercise 3 *Convince yourself that there is, in fact, a smallest set satisfying the condition stated in the definition above.* ⧫

In his paper [10], de Simone showed that each operation that can be defined using rules of the form (1) can be described by terms in the languages MEIJE [3] and SCCS [7]. (In case you are wondering what MEIJE and SCCS stand for, SCCS is an acronym for Synchronous Calculus of Communicating Systems. On the other hand, Meije is a mountain in the French Alps.) So, in this specific sense, those process calculi are expressively complete with respect to operations that can

be defined using rules of the form (1). In fact, achieving this result informed the design choices underlying the isolation of the set of basic operations in the language MEIJE.

Our concern here, however, is to use the rule formats we discuss to prove general results pertaining to the congruence properties of strong bisimilarity. Does every operation specified using rules in de Simone format preserve strong bisimilarity? We now proceed to provide a positive answer to this question.

Definition 3 Let σ and σ' be two closed substitutions. We write $\sigma \sim \sigma'$ when $\sigma(x) \sim \sigma'(x)$ for each $x \in V$. ♦

Theorem 1 Let (Σ, R) be a de Simone language. Then strong bisimilarity is a congruence—that is, $f(P_1, \ldots, P_n) \sim f(Q_1, \ldots, Q_n)$ holds, for each $f \in \Sigma$ of arity n, and terms P_i, Q_i such that $P_i \sim Q_i$ for each $1 \leq i \leq n$.

Proof: Let $\mathcal{R}$ be the smallest relation over the set of closed terms over Σ that satisfies the following clauses:

1. $\sim$ is included in $\mathcal{R}$, and
2. if $P_i \,\mathcal{R}\, Q_i$ for each $1 \leq i \leq n$ and $f \in \Sigma$ has arity n, then $f(P_1, \ldots, P_n) \,\mathcal{R}\, f(Q_1, \ldots, Q_n)$.

Note that, by the definition of $\mathcal{R}$, we have that $f(P_1, \ldots, P_n) \,\mathcal{R}\, f(Q_1, \ldots, Q_n)$ holds whenever $P_i \sim Q_i$ for each $1 \leq i \leq n$. (Why?) In fact,

$$\sigma(t) \,\mathcal{R}\, \sigma'(t) \tag{2}$$

holds for all substitutions σ and σ' such that $\sigma \sim \sigma'$ and for each term $t \in T(\Sigma)$. (Do not take my word for this claim; make sure you convince yourself that this property does hold!) Therefore, to prove the theorem it suffices only to show that the relation $\mathcal{R}$ defined above is a bisimulation. (See [2, Definition 3.2].) Since $\mathcal{R}$ is symmetric, our claim follows from the following statement:

> $(*)$ Assume that $P \,\mathcal{R}\, Q$ and $P \xrightarrow{a} P'$ for some P'. Then there is a Q' such that $Q \xrightarrow{a} Q'$ and $P' \,\mathcal{R}\, Q'$.

This we now proceed to prove by induction on the definition of $\mathcal{R}$.

If $P \,\mathcal{R}\, Q$ because $P \sim Q$, then the above statement follows immediately from the fact that $\sim$ is itself a bisimulation–see, e.g., [2, Theorem 3.1].

Assume therefore that $P \,\mathcal{R}\, Q$ because, for some operation f of arity n, and closed terms $P_1, \ldots, P_n, Q_1, \ldots, Q_n$,

1. $P = f(P_1, \ldots, P_n)$,
2. $Q = f(Q_1, \ldots, Q_n)$ and
3. $P_i \mathcal{R} Q_i$ for each $1 \leq i \leq n$.

Suppose, furthermore that $P \xrightarrow{a} P'$ for some P'. We shall show that there is a Q' such that $Q \xrightarrow{a} Q'$ and $P' \mathcal{R} Q'$. By item 3 above, in doing so, we may assume that property $(*)$ holds true for transitions of each P_i, $1 \leq i \leq n$.

Since the transition $P = f(P_1, \ldots, P_n) \xrightarrow{a} P'$ is provable from the rules in R, there are a rule r of the form (1) in R and a closed substitution σ such that

- $\sigma(x_i) = P_i$, for each $1 \leq i \leq n$,
- $\sigma(x_i) \xrightarrow{a_i} \sigma(y_i)$, for each $i \in I$, and
- $\sigma(t) = P'$.

Our order of business now will be to use the rule r to derive a matching transition from $Q = f(Q_1, \ldots, Q_n)$.

To this end, consider a transition $P_i = \sigma(x_i) \xrightarrow{a_i} \sigma(y_i)$, for some $i \in I$. Since $P_i \mathcal{R} Q_i$ and, by the inductive hypothesis, property $(*)$ holds true for transitions of each P_i, there is some Q'_i such that $Q_i \xrightarrow{a_i} Q'_i$ and $\sigma(y_i) \mathcal{R} Q'_i$. Note that we can find such a Q'_i for each premise of the rule.

Define now the closed substitution σ' as follows:

$$\sigma'(z) = \begin{cases} Q_i & \text{if } z = x_i \text{ for some } 1 \leq i \leq n, \\ Q'_i & \text{if } z = y_i \text{ for some } i \in I, \\ \sigma(z) & \text{otherwise.} \end{cases}$$

Since, for each $i \in I$, we have that

$$\sigma'(x_i) = Q_i \xrightarrow{a_i} Q'_i = \sigma'(y_i) \ ,$$

the substitution σ' satisfies all the premises of the rule. So, using the rule, we can infer the transition

$$Q = f(Q_1, \ldots, Q_n) \xrightarrow{a} \sigma'(t) \ .$$

Observe that, by construction, $\sigma \sim \sigma'$. Therefore, by (2), $P' = \sigma(t) \mathcal{R} \sigma'(t)$, and we are done. □

Note that, in the proof of the above theorem, we have not used at all the assumption that each variable occurs at most once in the target of a de Simone rule. It follows that the result we have just shown holds for a generalization of the de Simone format in which this constraint on the allowed rules is lifted. In what follows, we will see one further generalization of the theorem above. Before doing so, however, it is instructive to examine some generalizations of the the de Simone rule format for which Theorem 1 fails.

One of the constraints in the de Simone rule format is that all the variables appearing in rules be different. It is natural to ask oneself whether this requirement is necessary to prove Theorem 1. To see the need for the distinctness requirement for variables, let us add a binary operation eq to the language BPA, whose operational semantics is described by the rule below.

$$\frac{}{\mathrm{eq}(x,x) \stackrel{a}{\to} \delta}$$

This rule is not in de Simone format, and does not preserve strong bisimilarity. For example, you should have no problem in checking that a and $a + a$ are bisimilar. However, $\mathrm{eq}(a, a) \stackrel{a}{\to} \delta$, whereas $\mathrm{eq}(a, a+a)$ affords no transition. Intuitively, the problem with the above rule is that it allows one to check whether the two arguments of the eq operation are syntactically equal. This means that the behaviour of terms of the form $\mathrm{eq}(P, Q)$ is determined by the syntax of the arguments rather than by their behaviour.

Exercise 4 *Consider the language with the operations ω, ∞, f and g whose behaviour is defined by the following rules.*

$$\frac{}{\omega \stackrel{a}{\to} \omega} \qquad \frac{}{\infty \stackrel{a}{\to} \omega} \qquad \frac{x \stackrel{a}{\to} x}{f(x) \stackrel{a}{\to} \delta} \qquad \frac{x_1 \stackrel{a}{\to} y, \;\; x_2 \stackrel{a}{\to} y}{g(x_1, x_2) \stackrel{a}{\to} \delta}$$

Argue that neither f nor g preserves strong bisimilarity. ♦

Let us now turn our attention to the signature Σ_{Nat}. Intuitively, the constant Z stands for the number 0, the unary operation succ is the successor function over the natural numbers, and $\vee$ stands for the maximum of two numbers. Let us assume that the behaviour of a closed term t over this signature is to display a sequence of n consecutive a transitions, where n is the 'value' of the expression t. For example, we would expect the term succ(Z) to afford a single a-labelled

transition, and the term $\mathrm{succ}(\mathrm{Z}) \vee \mathrm{succ}(\mathrm{succ}(\mathrm{Z}))$ to afford two consecutive a-labelled transitions. For the succ operation, we can capture our intuition easily by means of the following axiom in de Simone format.

$$\frac{}{\mathrm{succ}(x) \xrightarrow{a} x}$$

Moreover, we expect that Z has no rule associated with it since that term has no outgoing transition. But what are the rules for the $\vee$ operation? One rule that immediately comes to mind is the following.

$$\frac{x_1 \xrightarrow{a} y_1, \;\; x_2 \xrightarrow{a} y_2}{x_1 \vee x_2 \xrightarrow{a} y_1 \vee y_2}$$

For instance, using this rule and the one for succ, we can prove the following transition:

$$\mathrm{succ}(\mathrm{Z}) \vee \mathrm{succ}(\mathrm{succ}(\mathrm{Z})) \xrightarrow{a} \mathrm{Z} \vee \mathrm{succ}(\mathrm{Z}) \;\; .$$

We would now expect that the target of the above transition, namely the term

$$\mathrm{Z} \vee \mathrm{succ}(\mathrm{Z}) \;\; ,$$

afford one more a-labelled transition and that it terminate in doing so. However, the only rule for the $\vee$ operation that we have at our disposal so far is not applicable to that term, since Z affords no transitions whatsoever. What we would like to have is a rule that states that a term $t \vee u$ also affords an a-labelled transition when one of the arguments of $\vee$ does so, and the other has 'terminated'. Since termination is characterized by the absence of a-labelled transitions, two natural rules capturing our intuition are

$$\frac{x_1 \xrightarrow{a} y_1, \;\; x_2 \xnrightarrow{a}}{x_1 \vee x_2 \xrightarrow{a} y_1} \qquad \frac{x_1 \xnrightarrow{a}, \;\; x_2 \xrightarrow{a} y_2}{x_1 \vee x_2 \xrightarrow{a} y_2} \;\; ,$$

where, for a term t, we write $t \xnrightarrow{a}$ to mean that t has no outgoing a-labelled transition. For instance, using the latter rule, we would be able to infer the transition

$$\mathrm{Z} \vee \mathrm{succ}(\mathrm{Z}) \xrightarrow{a} \mathrm{Z} \;\; .$$

The above set of rules gives a natural specification of the expected behaviour of terms over the signature Σ_{Nat}. However, the rules we have used are *not* in de Simone format, and use negative premises—that is, premises that refer to the

absence of transitions. As those of you who are familiar with the history of logic know well, negative premises must be handled with care as they easily lead to paradoxes when used in full generality. For instance, it is clear that a rule like

$$\frac{x \overset{a}{\not\to}}{x \overset{a}{\to} x}$$

is meaningless. However, negative premises often lead to natural language specifications, and can be 'well behaved' when their use is restricted appropriately. An example of a classic, and simple, rule format that uses negative premises is the GSOS format due to Bloom, Istrail and Meyer [5]. This we now proceed to introduce briefly.

Definition 4 Suppose Σ is a signature. A *GSOS rule* over Σ is a rule of the form:

$$\frac{\{x_i \overset{a_{ij}}{\to} y_{ij} \mid 1 \leq i \leq \ell,\ 1 \leq j \leq m_i\} \cup \{x_i \overset{b_{ik}}{\not\to} \mid 1 \leq k \leq n_i\}}{f(x_1, \ldots, x_\ell) \overset{c}{\to} t} \tag{3}$$

where all the variables are distinct, $m_i, n_i \geq 0$, f is an operation symbol from Σ with arity ℓ, and t is a term over Σ with variables including at most the x_i's and y_{ij}'s. (It need not contain all these variables.) The a_{ij}, b_{ik}, and c are actions. ♦

You should check that the rules we gave above for the operations in the signature Σ_{Nat} are all GSOS rules, and so is the rule for the operation a-and-b() given on page 2.

Definition 5 A *GSOS language* is a pair (Σ, R) where Σ is a finite signature and R is a finite set of GSOS rules over Σ. ♦

Informally, the intent of a GSOS rule is as follows. Suppose that we are wondering whether $f(P_1, \ldots, P_\ell)$ is capable of taking a c-step. We look at each rule for operation f and action c in turn. We inspect each positive premise $x_i \overset{a_{ij}}{\to} y_{ij}$, checking if P_i is capable of taking an a_{ij}-step for each j and if so calling the a_{ij}-children Q_{ij}. We also check the negative premises; if P_i is *in*capable of taking a b_{ik}-step for each k. If so, then the rule can be applied to derive the transition $f(P_1, \ldots, P_\ell) \overset{c}{\to} Q$, where Q is obtained from t by replacing each occurrence of x_i with P_i, and each occurrence of y_{ij} with Q_{ij}. This basically means that the transition relation associated with a GSOS language is the one defined by structural induction over closed terms using the rules. See [5] for the formal definition of the transition relation associated with a GSOS language.

Note that every de Simone language is also a GSOS language.

Does every operation specified using rules in GSOS format preserve strong bisimilarity? The answer is again positive, as stated in the following theorem.

Theorem 2 Let (Σ, R) be a GSOS language. Then strong bisimilarity is a congruence.

Exercise 5 *Prove the above theorem by modifying the proof of Theorem 1 appropriately.* ♦

The above theorem can be generalized to much more expressive rule formats. See, for instance, the survey papers [1, 9]. In those references, you will find a wealth of results that can be established by means of rule formats and many suggestions for further reading.

Remark 1 Those amongst you who are familiar with the operational semantics of CCS, as presented, e.g., on Table 2.2 on page 24 in [2], may have already realized that the SOS rule for process names is not in GSOS format, and therefore not in de Simone format either. That rule defines the transitions of process names, which are constants in the signature for CCS, thus:

$$\text{CON} \quad \frac{P \xrightarrow{\alpha} P'}{K \xrightarrow{\alpha} P'} \quad \text{where } K \stackrel{\text{def}}{=} P \ .$$

That rule is *not* in GSOS format since GSOS rules for constant symbols cannot have premises. It follows that rules of the form (3) cannot be used to describe the operational semantics of recursive definitions. (Note that, as witnessed by the rules in Exercise 4, one can easily specify non-terminating processes using de Simone rules.)

Rules like CON above can, however, be written in the so-called *tyft/tyxt format* introduced by Groote and Vaandrager in [6]. In fact, that format allows one to use rules of the form

$$\frac{\{t_i \xrightarrow{a_i} y_i \mid i \in I\}}{f(x_1, \ldots, x_n) \xrightarrow{a} t} \ ,$$

where

- I is an arbitrary index set,
- a and a_i, $i \in I$, are actions, and

- the variables $x_1, \ldots, x_n$ and y_i, $i \in I$, are all pairwise distinct, and
- t and t_i, $i \in I$, are terms.

You should be able to convince yourself readily that rule CON above fits this rule format.

As shown by Groote and Vaandrager, bisimilarity is preserved by operations whose behaviour is specified by rules in tyft/tyxt format, but the proof of this result is somewhat more involved than the one for Theorem 1 we presented above.

Rule formats that can handle binding operations, such $\lambda x.\_$ in the lambda calculus and input prefixing in value-passing CCS, are currently the subject of extensive work in the research community. See the survey paper [9] for an overview of current approaches and pointers to further literature. ♦

Acknowledgments I thank J. Srba for comments on a draft of this note.

References

[1] L. Aceto, W. Fokkink, and C. Verhoef. Structural operational semantics. In *Handbook of Process Algebra*, pages 197–292. North-Holland, 2001. Available from http://www.cs.vu.nl/~wanf/pubs/sos.pdf.

[2] L. Aceto, A. Ingólfsdóttir, K. G. Larsen, and J. Srba. *Reactive Systems: Modelling, Specification and Verification*. Cambridge University Press, August 2007.

[3] D. Austry and G. Boudol. Algèbre de processus et synchronisation. *Theoretical Comput. Sci.*, 30:91–131, 1984.

[4] J.C.M. Baeten and W. P. Weijland. *Process Algebra*. Cambridge University Press, 1990.

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

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

[7] R. Milner. Calculi for synchrony and asynchrony. *Theoretical Comput. Sci.*, 25:267–310, 1983.

[8] R. Milner. *Communication and Concurrency*, Prentice-Hall International, Englewood Cliffs.

[9] M.R. Mousavi, M. A. Reniers, J.F. Groote. SOS Formats and Meta-Theory: 20 Years After. *Theoretical Comput. Sci.*, 373(3):238–272, 2007. Available from http://www.win.tue.nl/~mousavi/tcs07.pdf.

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