Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
Electronic Notes in Theoretical Computer Science 265 (2010) 145–160
www.elsevier.com/locate/entcs

# On Rule Formats for Zero and Unit Elements

Luca Aceto[1] Matteo Cimini[1] Anna Ingolfsdottir[1]

*ICE-TCS, School of Computer Science*
*Reykjavik University*
*Menntavegur 1, IS 101 Reykjavik, Iceland*

MohammadReza Mousavi Michel A. Reniers[2]

*Department of Computer Science*
*Eindhoven University of Technology*
*P.O. Box 513, NL-5600 MB Eindhoven, The Netherlands*

**Abstract**

This paper proposes a rule format for Structural Operational Semantics guaranteeing that certain constants act as left or right zero elements for a set of binary operators. Our design approach is also applied to reformulate an earlier rule format for unit elements developed by some of the authors. Examples of left and right zero, as well as unit, elements from the literature are shown to be checkable using the provided formats.

*Keywords:* Structural Operational Semantics (SOS), GSOS format, bisimulation equivalence, zero element, unit element

## 1 Introduction

In the last three decades, Structural Operational Semantics (SOS), see, e.g., [4,17,19,20], has been shown to be a powerful way to specify the semantics of programming and specification languages. In this approach to semantics, languages can be given a clear behaviour in terms of states and transitions, where the collection of transitions is specified by means of a set of syntax-driven inference rules. Based on this semantics in terms of state transitions, we often want to prove general algebraic laws about the languages, which describe semantic properties of the various operators they involve modulo a notion of behavioural equivalence or preorder of interest. For example, the reader may think about the field of process

[1] The work of Aceto, Cimini and Ingolfsdottir has been partially supported by the projects 'New Developments in Operational Semantics' (nr. 080039021) and 'Meta-theory of Algebraic Process Theories' (nr. 100014021) of the Icelandic Research Fund.

[2] Email: m.a.reniers@tue.nl


doi:10.1016/j.entcs.2010.08.009

algebra, where it is important to check whether certain operators are, say, commutative and associative with respect to bisimilarity.

This paper aims at contributing to an ongoing line of research whose goal is to ensure the validity of algebraic properties *by design*, using the so called *SOS rule formats* [5]. Results in this research area roughly state that if the specification of (parts of) the operational semantics of a language has a certain form then some semantic property is guaranteed to hold. The literature on SOS provides rule formats for basic algebraic properties of operators such as commutativity [16], associativity [12] and idempotence [1]. The main advantage of this approach is that one is able to verify the desired property by syntactic checks that can be mechanized. Moreover, it is interesting to develop rule formats for establishing semantic properties since results so obtained apply to a broad class of languages.

Recently, some of the authors provided in [6] a rule format guaranteeing another basic algebraic property not addressed before: the existence of left and right unit elements for operators. In the present paper, we follow the work presented in [6] and we develop a rule format guaranteeing instead that certain constants act as left or right zero elements for a set of binary operators. Namely, a function $f$ has a left (respectively, right) zero element $c$, modulo some notion of behavioural equivalence, whenever the equation $f(c, x) = c$ (respectively, $f(x, c) = c$) holds. A constant $c$ satisfying the above equation(s) is also said to be *absorbing* for the operator $f$.

A classic example of a left zero element within the realm of process algebra is provided by the constant $\delta$, for deadlock, from BPA [9], which satisfies the laws:

$$\delta \cdot x = \delta \quad \text{and} \quad \delta \between x = \delta \ ,$$

where '$\cdot$' and '$\between$' stand for sequential composition and left merge, respectively.

In this paper, we formulate our zero-element format within the GSOS languages of Bloom, Istrail and Meyer [11]. In particular, we benefit from the logic of transition formulae developed by some of the authors in [2], which is tailored for reasoning about the satisfiability of premises of GSOS rules. (The full version of the paper [3] offers also a syntactic rule format for left and right zero elements that applies to SOS rules that are more general than GSOS ones.)

The final part of the paper is devoted to applying the design ideas underlying the GSOS-based format for left and right zero elements to reformulate the format for left and right unit elements from [6]. The resulting format turns out to be incomparable in power to the original one, but it is expressive enough to check all the examples discussed in [6].

Mechanizing the rule formats in a tool-set is a long-term goal of research on SOS rule formats. We believe that the GSOS-based rule formats we present in this paper are strong candidates for mechanization insofar as zero and unit elements are concerned.

**Roadmap of the paper**

Section 2 repeats some standard definitions from the theory of SOS and from the logic of initial transitions from [2]. Section 3 provides the format for left and right zero elements and Section 4 shows how several examples of left and right zero elements from the literature fit the format. In Section 5 we provide a rule format for unit elements adapting the ideas from Section 3. We conclude the paper in Section 6 with an overview of its main contributions and with a mention of further results that may be found in the full version of the paper [3].

## 2 Preliminaries

In this section we recall some standard definitions from the theory of SOS. We refer the readers to, e.g., [4] and [17] for more information.

### *2.1 Transition system specifications and bisimilarity*

**Definition 2.1** [Signatures, terms and substitutions] We let $V$ denote an infinite set of variables and use $x, x', x_i, y, y', y_i, \ldots$ to range over elements of $V$. A *signature* $\Sigma$ 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 $\Sigma$ 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 $s, t$, possibly subscripted and/or superscripted, 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 $p, q, p', p_i, \ldots$ to range over closed terms. A *substitution* $\sigma$ is a function of type $V \to \mathbb{T}(\Sigma)$. We extend the domain of substitutions to terms homomorphically and write $\sigma(t)$ for the result of applying the substitution $\sigma$ to the term $t$. If the range of a substitution lies in $\mathbb{C}(\Sigma)$, we say that it is a *closed substitution*.

**Definition 2.2** [Transition system specification] A *transition system specification* (TSS) is a triple $(\Sigma, \mathcal{L}, D)$ where

- $\Sigma$ is a signature.
- $\mathcal{L}$ is a set of labels (or actions) ranged over by $a, b, l$. If $l \in \mathcal{L}$, and $t, t' \in \mathbb{T}(\Sigma)$ we say that $t \xrightarrow{l} t'$ is a *positive transition formula* and $t \xnrightarrow{l}$ is a *negative transition formula*. A transition formula (or just formula), typically denoted by $\phi$ or $\psi$, is either a negative transition formula or a positive one.
- $D$ is a set of *deduction rules*, i.e., tuples of the form $(\Phi, \phi)$ where $\Phi$ is a set of formulae and $\phi$ is a positive formula. We call the formulae contained in $\Phi$ the *premises* of the rule and $\phi$ the *conclusion*.

We write $vars(r)$ to denote the set of variables appearing in a deduction rule $r$. We say that a formula or a deduction rule is *closed* if all of its terms are closed. Substitutions are also extended to formulae and sets of formulae in the natural way. For a rule $r$ and a substitution $\sigma$, the rule $\sigma(r)$ is called a substitution instance of $r$. A set of positive closed formulae is called a *transition relation*.

We often refer to a positive transition formula $t \xrightarrow{l} t'$ as a *transition* with $t$ being its *source*, $l$ its *label*, and $t'$ its *target*. A deduction rule $(\Phi, \phi)$ is typically written as $\frac{\Phi}{\phi}$. An *axiom* is a deduction rule with an empty set of premises. We call a deduction rule *$f$-defining* when the outermost function symbol appearing in the source of its conclusion is $f$.

*In this paper, for each constant $c$, we assume that each $c$-defining deduction rule is an axiom of the form $c \xrightarrow{l} p$ for some label $l$ and closed term $p$.* This is not a real restriction since all practical cases we know of do actually satisfy this property. For GSOS languages [11], whose definition is given below and that will be our focus in the remainder of this study, this restriction is automatically satisfied.

**Definition 2.3** [GSOS rule] Suppose $\Sigma$ is a signature. A *GSOS rule* $r$ over $\Sigma$ is a rule of the form:

$$\frac{\bigcup_{i=1}^{l} \left\{ x_i \xrightarrow{a_{ij}} y_{ij} | 1 \leq j \leq m_i \right\} \quad \cup \quad \bigcup_{i=1}^{l} \left\{ x_i \overset{b_{ik}}{\not\rightarrow} | 1 \leq k \leq n_i \right\}}{f(x_1, \ldots, x_l) \xrightarrow{c} t} \tag{1}$$

where all the variables are distinct, $m_i, n_i \geq 0$, $a_{ij}$, $b_{ik}$, and $c$ are actions from a finite set, $f$ is a function symbol from $\Sigma$ with arity $l$, and $t$ is a term in $\mathbb{T}(\Sigma)$ such that $vars(t) \subseteq \{x_1, \ldots, x_l\} \cup \{y_{ij} \mid 1 \leq i \leq l, 1 \leq j \leq m_i\}$.

**Definition 2.4** A *GSOS language* is a triple $G = (\Sigma_G, \mathcal{L}, R_G)$, where $\Sigma_G$ is a finite signature, $\mathcal{L}$ is a finite set of action labels and $R_G$ is a finite set of GSOS rules over $\Sigma_G$. The transition relation $\rightarrow_G$ associated with a GSOS language $G$ is the one defined by the rules using structural induction over closed $\Sigma_G$-terms.

**Definition 2.5 (Bisimulation and bisimilarity [15,18])** *Let $G = (\Sigma_G, \mathcal{L}, R_G)$ be a GSOS language. A relation $\mathcal{R} \subseteq \mathbb{C}(\Sigma_G) \times \mathbb{C}(\Sigma_G)$ is a* bisimulation relation *if and only if $\mathcal{R}$ is symmetric and, for all $p_0, p_1, p_0' \in \mathbb{C}(\Sigma_G)$ and $l \in \mathcal{L}$,*

$$(p_0 \,\mathcal{R}\, p_1 \wedge p_0 \xrightarrow{l}_G p_0') \Rightarrow \exists p_1' \in \mathbb{C}(\Sigma_G).\ (p_1 \xrightarrow{l}_G p_1' \wedge p_0' \,\mathcal{R}\, p_1').$$

*Two terms $p_0, p_1 \in \mathbb{C}(\Sigma_G)$ are called* bisimilar*, denoted by $p_0 \leftrightarrows p_1$, when there exists a bisimulation relation $\mathcal{R}$ such that $p_0 \,\mathcal{R}\, p_1$.*

Bisimilarity is extended to open terms by requiring that $s, t \in \mathbb{T}(\Sigma)$ are bisimilar when $\sigma(s) \leftrightarrows \sigma(t)$ for each closed substitution $\sigma : V \to \mathbb{C}(\Sigma_G)$.

### 2.2 The logic of initial transitions

In this section, for the sake of completeness, we discuss the logic we employ in the definition of our rule format for left and right zero elements based on GSOS. The *logic of initial transitions* has been recently introduced by some of the authors in [2] in order to reason about the satisfiability of the premises of GSOS rules. The set of initial transition formulae over a finite set of actions $\mathcal{L}$ is defined by the following grammar, where $a \in \mathcal{L}$:

$$F ::= \quad \mathsf{True} \mid x \xrightarrow{a} \quad \mid \neg F \mid F \wedge F \ .$$

As usual, we write $\mathsf{False}$ for $\neg\mathsf{True}$, and $F \vee F'$ for $\neg(\neg F \wedge \neg F')$.

The semantics of this logic is given by a satisfaction relation $\models$ that is defined, relative to a GSOS language $G = (\Sigma_G, \mathcal{L}, R_G)$, by structural recursion on $F$ in the following way, where $\sigma$ is a closed substitution and $\rightarrow_G$ is the collection of transitions that can be proven using the rules in $R_G$:

$$\begin{aligned}
&\rightarrow_G, \sigma \models \mathsf{True} \quad \text{always}\\
&\rightarrow_G, \sigma \models x \xrightarrow{a} \Leftrightarrow \sigma(x) \xrightarrow{a}_G p, \text{ for some } p\\
&\rightarrow_G, \sigma \models \neg F \Leftrightarrow \text{not } \rightarrow_G, \sigma \models F\\
&\rightarrow_G, \sigma \models F \wedge F' \Leftrightarrow \rightarrow_G, \sigma \models F \text{ and } \rightarrow_G, \sigma \models F' \ .
\end{aligned}$$

The reader familiar with Hennessy-Milner logic [13] will have noticed that the propositions of the form $x \xrightarrow{a}$ correspond to Hennessy-Milner formulae of the form $\langle a \rangle \mathsf{True}$. In what follows, we consider formulae up to commutativity and associativity of $\wedge$.

We use the logic to turn the set of premises $\Phi$ of a GSOS rule into a formula that describes the collection of closed substitutions that satisfy $\Phi$. The conversion procedure $\mathsf{hyps}$ is borrowed from [2]. Formally,

$$\begin{aligned}
\mathsf{hyps}(\emptyset) &= \mathsf{True}\\
\mathsf{hyps}(\{x \xrightarrow{a} y\} \cup \Phi) &= (x \xrightarrow{a}) \wedge \mathsf{hyps}(\Phi \setminus \{x \xrightarrow{a} y\})\\
\mathsf{hyps}(\{x \xrightarrow{a} \!\!\!\!\!\!\rightarrow\ \} \cup \Phi) &= \neg(x \xrightarrow{a}) \wedge \mathsf{hyps}(\Phi \setminus \{x \xrightarrow{a} \!\!\!\!\!\!\rightarrow\ \}) \ .
\end{aligned}$$

Intuitively, if $\Phi$ is the set of premises of a rule then $\mathsf{hyps}(\Phi)$ is the conjunction of the corresponding initial transition formulae. For example,

$$\mathsf{hyps}(\{x \xrightarrow{a} y, z \xrightarrow{b} \!\!\!\!\!\!\rightarrow\ \}) = (x \xrightarrow{a}) \wedge \neg(z \xrightarrow{b}) \ .$$

If $J$ is a finite set of GSOS rules, we overload $\mathsf{hyps}$ and write:

$$\mathsf{hyps}(J) = \bigvee_{r \in J} \mathsf{hyps}(\Phi_r) \ ,$$

where $\Phi_r$ is the set of premises of rule $r$.

We write $\models_G F \Rightarrow F'$ iff every substitution that satisfies $F$ also satisfies $F'$. This semantic entailment preorder is decidable, as shown in [2].

**Theorem 2.6 (Decidability of entailment)** *Let $G$ be a GSOS language. Then, for all formulae $F$ and $F'$, it is decidable whether $\models_G F \Rightarrow F'$ holds.*

As a matter of fact, when $\Phi$ is the set of the premises of a rule $r$, checking whether $\models_G \mathsf{True} \Rightarrow \mathsf{hyps}(\Phi)$ holds is equivalent to checking whether the rule $r$ is always firable. Conversely, checking whether $\models_G \mathsf{hyps}(\Phi) \Rightarrow \mathsf{False}$ holds is equivalent to checking whether the rule $r$ never fires. These considerations will be useful in the remainder of the paper. Our definition of the rule format for left and right zero elements makes use of the logic and especially of these two kinds of entailment. The semantic entailment is, moreover, used in a simplified fashion where one does not need to check all the closed substitutions, but only those that map one variable to the left or right zero element constant under consideration. We now proceed to formalize this notion.

**Definition 2.7** Let $G = (\Sigma_G, \mathcal{L}, R_G)$ be a GSOS language. For each formula $F$, constant $c \in \Sigma_G$ and variable $x$, we define the formula $F[x \mapsto c]$ by structural recursion on $F$ as follows:

$$\begin{aligned}
\mathsf{True}[x \mapsto c] &= \mathsf{True} \\
(x \xrightarrow{a} )[x \mapsto c] &= \begin{cases} \mathsf{True} & \text{if there is a } c\text{-defining axiom } c \xrightarrow{a} p \text{ for some } p \\ \mathsf{False} & \text{otherwise} \end{cases} \\
(y \xrightarrow{a} )[x \mapsto c] &= y \xrightarrow{a} \qquad \text{if } x \neq y \\
(\neg F)[x \mapsto c] &= \neg(F[x \mapsto c]) \\
(F_1 \wedge F_2)[x \mapsto c] &= (F_1[x \mapsto c]) \wedge (F_2[x \mapsto c]) \ .
\end{aligned}$$

The connection between $F$ and $F[x \mapsto c]$ is provided by the following lemma.

**Lemma 2.8** *Let $G = (\Sigma_G, \mathcal{L}, R_G)$ be a GSOS language. Let $F$ be a formula, $c$ be a constant in $\Sigma_G$ and $x$ be a variable. Then, for each closed substitution $\sigma$,*

$$\rightarrow_G, \sigma \models F[x \mapsto c] \quad \textit{iff} \quad \rightarrow_G, \sigma[x \mapsto c] \models F \ ,$$

*where $\sigma[x \mapsto c]$ denotes the substitution that maps $x$ to $c$ and acts like $\sigma$ on all the other variables.*

As a consequence of the above lemma, checking whether $F$ holds for all substitutions that map variable $x$ to a constant $c$ amounts to showing that the formula $F[x \mapsto c]$ is satisfied by all substitutions—that is, showing that $F[x \mapsto c]$ is a tautology over $G$.

## 3 Rule format for zero elements

In this section we provide a rule format guaranteeing that certain constants act as left or right zero elements for a set of binary operators. To this end we employ a variation on the technique developed by some of the authors in [6] for left or right unit elements.

As in [6], we make use of an equivalence relation between terms called *zero-context equivalence*, which is the counterpart of the unit-context equivalence from [6]. Intuitively if $c$ is a left zero element for an operator $f$ and $c$ is also a

right zero element for $g$, then the terms $f(c, t_1)$ and $g(t_2, c)$ are both zero-context equivalent to $c$ and zero-context equivalent to each other.

In the following formal definition of zero-context equivalence, it is useful to consider $(f, c) \in L$ as stating that '$c$ acts as a left zero element for the operator $f$' and analogously $(f, c) \in R$ indicates that the constant $c$ is a right zero element for $f$.

**Definition 3.1** [Zero-context equivalent terms] Given sets $L, R \subseteq \Sigma \times \Sigma$ of pairs of binary function symbols and constants, $\overset{L,R}{\cong}_0$ is the smallest equivalence relation satisfying the following constraints, for each $s \in \mathbb{T}(\Sigma)$:

(i) $\forall (f, c) \in L.\ c \overset{L,R}{\cong}_0 f(c, s)$, and

(ii) $\forall (g, d) \in R.\ d \overset{L,R}{\cong}_0 g(s, d)$.

We say that two terms $s, t \in \mathbb{T}(\Sigma)$ are *zero-context equivalent*, if $s \overset{L,R}{\cong}_0 t$.

Since the sets $L$ and $R$ are always clear from the context, in the remainder of the paper we write $\cong_0$ in place of $\overset{L,R}{\cong}_0$.

**Theorem 3.2 (Decidability of zero-context equivalence)** *Let $L, R \subseteq \Sigma \times \Sigma$ be finite sets of pairs of binary function symbols and constants. Then, for all terms $t, u \in \mathbb{T}(\Sigma)$, it is decidable whether $t \overset{L,R}{\cong}_0 u$ holds.*

In order to remain in line with the terminology in [6], in the following definition we talk about left- and right-aligned pairs. The conditions of our format will not try to ensure firability/unfirability of rules by syntactic means as in the rule format for unit elements from [6], but they instead exploit the logic of initial transition formulae to incorporate a modicum of semantic reasoning within the rule format.

**Definition 3.3** [Left- and right-aligned pairs] Let $G$ be a GSOS language. The sets $L$ and $R$ of pairs of binary function symbols and constants are the largest sets satisfying the following constraints.

**1.** For each $(f, c) \in L$, the following conditions hold.

**a.** For each axiom $c \xrightarrow{a} t$, there exists a set $J$ of rules of the form

$$\frac{\Phi}{f(x_0, x_1) \xrightarrow{a} t'}$$

such that

**i.** $\models_G \mathsf{True} \Rightarrow \mathsf{hyps}(J)[x_0 \mapsto c]$, and

**ii.** for each rule in $J$, one of the following cases holds:

**A.** there is some variable $y \in vars(t')$ such that $x_0 \xrightarrow{a} y \in \Phi$ and $\sigma(t') \cong_0 t$, where $\sigma$ is the substitution mapping $x_0$ to $c$, $y$ to $t$ and is the identity on all the other variables, or

**B.** $\sigma(t') \cong_0 t$, where $\sigma$ is the substitution mapping $x_0$ to $c$ and is the identity on all the other variables.

**b.** For each $f$-defining deduction rule

$$\frac{\Phi}{f(x_0, x_1) \xrightarrow{a} t'}$$

one of the following cases holds:

**i.** there exists an axiom $c \xrightarrow{a} t$ such that

**A.** there is some variable $y \in vars(t')$ such that $x_0 \xrightarrow{a} y \in \Phi$ and $\sigma(t') \cong_0 t$, where $\sigma$ is the substitution mapping $x_0$ to $c$, $y$ to $t$ and is the identity on all the other variables, or

**B.** $\sigma(t') \cong_0 t$, where $\sigma$ is the substitution mapping $x_0$ to $c$ and is the identity on all the other variables.

**ii.** $\models_G \mathsf{hyps}(\Phi)[x_0 \mapsto c] \Rightarrow \mathsf{False}$.

**2.** The definition of right-aligned pairs of operators and constant symbols—that is, those such that $(f, c) \in R$—is symmetric and is not repeated here.

For a function symbol $f$ and a constant $c$, we call $(f, c)$ *left aligned* (respectively, *right aligned*) if $(f, c) \in L$ (respectively, $(f, c) \in R$).

Let $G$ be a GSOS language over a signature including at least one constant. Since $\mathsf{hyps}(J)$ is a disjunctive formula, condition 1.a.i. in the above definition implies that the set $J$ is non-empty. On the other hand, condition 1.b.ii. says that the premises of the rule under consideration cannot be satisfied by any closed substitution that maps the variable $x_0$ to the constant $c$.

In condition 1.a. and its symmetric counterpart, one must identify a *set* $J$ of rules. To understand why, the reader should consider the following TSS with constants $\mathbf{0}$ (with no rule) and $\mathrm{RUN}_a$, and a function symbol $f$ defined as follows

$$\frac{}{\mathrm{RUN}_a \xrightarrow{a} \mathrm{RUN}_a} \qquad (y)\ \frac{x \xrightarrow{a} x' \quad y \xrightarrow{a} y'}{f(x, y) \xrightarrow{a} x'} \qquad (not\text{–}y)\ \frac{x \xrightarrow{a} x' \quad y \not\xrightarrow{a}}{f(x, y) \xrightarrow{a} x'}\ .$$

The rules $(y)$ and $(not\text{–}y)$ only together allow the operator $f$ to simulate the behaviour of the constant $\mathrm{RUN}_a$: no matter what closed term is substituted for the argument variable $y$, we are sure that one of the two rules fires and that the transition leads to $\mathrm{RUN}_a$. In Definition 3.3, these two properties are guaranteed, respectively, by conditions 1.a.i. and 1.a.ii.

**Theorem 3.4** *Let $G$ be a GSOS language. Assume that $L$ and $R$ are the sets of left- and right-aligned function symbols according to Definition 3.3. For each $(f, c) \in L$, it holds that $f(c, x) \leftrightarrows c$. Symmetrically, for each $(f, c) \in R$, it holds that $f(x, c) \leftrightarrows c$.*

The following result is a consequence of Theorems 2.6 and 3.2.

**Theorem 3.5** *For GSOS languages, the sets $L$ and $R$ can be effectively constructed.*

We conclude this section by discussing some of the constraints in Definition 3.3

in order to argue that they cannot be easily relaxed. In what follows, we focus on the conditions that left-aligned pairs must meet. First of all, note that relaxing the constraint of GSOS rules that $x_0 \not\equiv x_1$ would jeopardize Theorem 3.4. To see this, consider the TSS with constant $\mathrm{RUN}_a$ and binary operator $f$ with rule

$$\frac{x_0 \xrightarrow{a} y_0}{f(x_0, x_0) \xrightarrow{a} y_0} \ .$$

It is not hard to check that $L = \{(f, \mathrm{RUN}_a)\}$ and $R = \emptyset$ satisfy all the other constraints of GSOS rules and Definition 3.3. Due to the presence of axiom $\mathrm{RUN}_a \xrightarrow{a} \mathrm{RUN}_a$ constraint 1.b.i.A. is met in this case. However, $\mathrm{RUN}_a$ is *not* a left zero element for $f$. For example, the term $f(\mathrm{RUN}_a, f(\mathrm{RUN}_a, \mathrm{RUN}_a))$ affords no transition and therefore cannot be bisimilar to $\mathrm{RUN}_a$.

The following example shows that relaxing the GSOS requirement that $x_1 \notin \{y_{ij} \mid 1 \leq i \leq l, 1 \leq j \leq m_i\}$ would also invalidate Theorem 3.4. To see this, consider the TSS with constant $\mathrm{RUN}_a$ and binary operator $f$ with rule

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

Again, it is not hard to check that $L = \{(f, \mathrm{RUN}_a)\}$ and $R = \emptyset$ satisfy all the other constraints of GSOS rules and Definition 3.3. However, $f(\mathrm{RUN}_a, f(\mathrm{RUN}_a, \mathrm{RUN}_a))$ affords no transition and therefore cannot be bisimilar to $\mathrm{RUN}_a$. This means that $\mathrm{RUN}_a$ is *not* a left zero element for $f$.

The role played by requirements 1.a.i. and 1.a.ii. in ensuring that, modulo bisimilarity, $f(c, p)$ affords 'the same transitions as $c$', for each $p$, is highlighted by the following two examples.

**Example 3.6** Consider the TSS with constants $\mathbf{0}$ and $a\&b$, and a binary operator $f$ with rules:

$$\frac{}{a\&b \xrightarrow{a} \mathbf{0}} \qquad \frac{}{a\&b \xrightarrow{b} \mathbf{0}} \qquad \frac{x_0 \xrightarrow{b} \quad x_0 \xrightarrow{a} y_0}{f(x_0, x_1) \xrightarrow{a} y_0} \qquad \frac{x_0 \xrightarrow{a} \quad x_0 \xrightarrow{b} y_0}{f(x_0, x_1) \xrightarrow{b} y_0} \ .$$

It is not hard to check that $L = \{(f, a\&b)\}$ and $R = \emptyset$ satisfy all the constraints in Definition 3.3 apart from 1.a.i. In particular, any singleton set $J$ of $f$-defining deduction rules satisfies constraint 1.a.ii.A. (and thus constraint 1.a.). However, the term $f(a\&b, \mathbf{0})$ affords no transition unlike $a\&b$. Therefore $a\&b$ is not a left zero element for $f$. ⊠

**Example 3.7** Consider the TSS with constants $\mathrm{RUN}_a$, $\mathrm{RUN}_b$ and $c$ and a binary operator $f$ with rules:

$$\frac{}{c \xrightarrow{a} \mathrm{RUN}_a} \qquad \frac{}{c \xrightarrow{a} \mathrm{RUN}_b} \qquad \frac{x_1 \xrightarrow{a} y_1}{f(x_0, x_1) \xrightarrow{a} \mathrm{RUN}_a} \qquad \frac{x_1 \xrightarrow{b} y_1}{f(x_0, x_1) \xrightarrow{a} \mathrm{RUN}_b}$$

Let $L = \{(f, c)\}$ and $R$ be empty. We claim that all the conditions in Definition 3.3 are met, apart from 1.a.ii. To see this, note, first of all, that each closed term in this language initially affords an $a$-labelled or a $b$-labelled transition. Therefore, the formula $x_1 \xrightarrow{a} \vee\, x_1 \xrightarrow{b}$ is a tautology. It follows that condition 1.a.i. can be met for both the $c$-defining axioms by taking $J$ to contain both $f$-defining rules. Observe that condition 1.a.ii. fails for this $J$, but condition 1.b.i.B. is met for both $f$-defining rules by matching the first $f$-defining rule with the first rule for $c$ and the second $f$-defining rule with the second rule for $c$.

However, $c$ is *not* a left zero element for $f$. For example, $f(c, \mathrm{RUN}_a)$ only affords an $a$-labelled transition to $\mathrm{RUN}_a$ and therefore cannot match the transition $c \xrightarrow{a} \mathrm{RUN}_b$. ⊠

As witnessed, e.g., by Example 4.3 to follow, constraint 1.b.i. enhances the generality of our format. Indeed, if we removed constraint 1.b.i. and a left-aligned pair $(f, c)$ satisfied condition 1.b.ii., then no rule for $f$ would be applicable to a closed term of the form $f(c, p)$. Therefore, no term of the form $f(c, p)$ would afford a transition. Since $(f, c)$ satisfies condition 1.a. in Definition 3.3, the collection of $c$-defining axioms must be empty. As a consequence, the resulting format would be unable to handle left zero elements such as $\mathrm{RUN}_a$ that afford some transition. Examples of constants with deduction axioms in the literature are immediate deadlock [8], which acts as a left zero element for sequential composition, parallel composition, left merge and communication merge and as a right zero element for parallel composition and communication merge, and delayable deadlock from [7], which is a left zero element for sequential composition.

## 4 Examples

In this section we show that several examples of zero elements from the literature indeed fit the format described in Section 3.

**Example 4.1** [Synchronous parallel composition] Consider the synchronous parallel composition from CSP [14] over a set of actions $\mathcal{L}$ with rules:

$$\frac{x \xrightarrow{a} x' \quad y \xrightarrow{a} y'}{x \parallel_{\mathcal{L}} y \xrightarrow{a} x' \parallel_{\mathcal{L}} y'} \quad (a \in \mathcal{L}) \ .$$

We know that the inaction constant $\mathbf{0}$, with no rules, is a left and right zero element for $\parallel_{\mathcal{L}}$. Let $L = R = \{(\parallel_{\mathcal{L}}, \mathbf{0})\}$. Since the constant $\mathbf{0}$ has no axioms, condition 1.a. is vacuously satisfied. In order to see that also condition 1.b. is satisfied, it is sufficient to notice that the only rule for $\parallel_{\mathcal{L}}$ can never fire because $\mathbf{0}$ has no transitions.

Indeed, the entailment $\models_G (x \xrightarrow{a} \wedge\, y \xrightarrow{a})[x \mapsto \mathbf{0}] \Rightarrow \mathsf{False}$ holds and condition 1.b.ii. is met. The symmetric counterpart of clause 1.b. is handled in similar fashion. The well-known laws

$$\mathbf{0} \parallel_{\mathcal{L}} y \leftrightarrows \mathbf{0} \quad \text{and} \quad x \parallel_{\mathcal{L}} \mathbf{0} \leftrightarrows \mathbf{0}$$

thus follow from Theorem 3.4. ⊠

**Example 4.2** [Left merge operator] Consider the left merge operator from [9].

$$\frac{x \xrightarrow{a} x'}{x \between y \xrightarrow{a} x' \parallel y}$$

Here $\parallel$ stands for the merge operator from [9], whose SOS specification is immaterial for this example; see Example 4.3 to follow. Let $L = \{(\between, \mathbf{0})\}$ and $R = \emptyset$. We claim that $L$ meets the constraints in Definition 3.3. It is easy to check that the claim is true by the same reasoning used in Example 4.1. This time it is sufficient to check condition 1. because $\mathbf{0}$ is just a *left* zero element for $\between$. By Theorem 3.4 the validity of the law $\mathbf{0} \between y \leftrightarrows \mathbf{0}$ follows. Note that the pair $\{(\between, \mathbf{0})\}$ cannot be added to $R$ because the symmetric version of condition 1.b. would be violated. Indeed $\mathbf{0}$ is *not* a right zero element for $\between$. ⊠

**Example 4.3** [Merge operator] Let $\mathcal{L}$ be the set of actions. Consider the classic merge operator $\parallel$ with the following rules, where $a \in \mathcal{L}$.

$$\frac{x \xrightarrow{a} x'}{x \parallel y \xrightarrow{a} x' \parallel y} \qquad \frac{y \xrightarrow{a} y'}{x \parallel y \xrightarrow{a} x \parallel y'}$$

Let $\mathrm{RUN}_{\mathcal{L}}$ be a constant defined by axioms $\mathrm{RUN}_{\mathcal{L}} \xrightarrow{a} \mathrm{RUN}_{\mathcal{L}}$ for each action $a \in \mathcal{L}$. We claim that the constant $\mathrm{RUN}_{\mathcal{L}}$ is both a left and right zero element for $\parallel$. This can be checked using Theorem 3.4. Indeed, let $L = R = \{(\parallel, \mathrm{RUN}_{\mathcal{L}})\}$. It is easy to see that condition 1.a. in Definition 3.3 is met for $\mathrm{RUN}_{\mathcal{L}} \xrightarrow{a} \mathrm{RUN}_{\mathcal{L}}$ by taking the instance of the left-hand rule for $\parallel$ with action $a$. Condition 1.b. is for the left-hand deduction rule met via constraint 1.b.i.A. due to the presence of the axiom for $\mathrm{RUN}_{\mathcal{L}}$ for action $a$. For the right-hand rule for $\parallel$ with action $a$, condition 1.b.i.B is met since

$$\sigma(x \parallel y') \equiv \mathrm{RUN}_{\mathcal{L}} \parallel \sigma(y') \cong_0 \mathrm{RUN}_{\mathcal{L}}$$

for a substitution $\sigma$ with $\sigma(x) = \mathrm{RUN}_{\mathcal{L}}$ and that acts as the identity function otherwise, and $\mathrm{RUN}_{\mathcal{L}} \xrightarrow{a} \mathrm{RUN}_{\mathcal{L}}$ is one of the axioms for the constant $\mathrm{RUN}_{\mathcal{L}}$. ⊠

**Example 4.4** [A right-choice operator] In this example we apply our format to a non-standard operator. For the sake of simplicity we assume that $a$ is the only action. Consider a variant of the choice operator of Milner's CCS [15], where the right-hand argument has a higher priority than the left-hand argument, i.e., the scheduler executes the left-hand argument only when the other one has no transitions. The rules for such an operator are as follows:

$$\frac{x \xrightarrow{a} x' \quad y \overset{a}{\not\to}}{x \leftarrow\!\!\!+ y \xrightarrow{a} x'} \qquad \frac{y \xrightarrow{a} y'}{x \leftarrow\!\!\!+ y \xrightarrow{a} y'} \ .$$

Let $c$ be any constant whose behaviour is defined by a non-empty, finite collection of axioms $\{c \xrightarrow{a} p_i \mid i \in I\}$, where $I$ is some index set. Reasoning as in the previous examples, using Theorem 3.4, we are able to prove the validity of the law $x \leftarrow\!\!\!+ c \leftrightarrows c$. We leave the details to the reader. The operator studied in this example bears resemblance with the preferential choice operator $+\!\!\!\rightarrow$ from [10]. ⊠

## 5 From zero to unit

In this section we reformulate the unit element format of [6] following the lines of Definition 3.3. For the sake of clarity and completeness we repeat here the definition of unit-context equivalence from [6].

**Definition 5.1** [Unit-context equivalence [6]] Given sets $L, R \subseteq \Sigma \times \Sigma$ of pairs of binary function symbols and constants, $\overset{L,R}{\cong}$ is the smallest equivalence relation satisfying the following constraints, for each $s \in \mathbb{T}(\Sigma)$:

(i) $\forall (f, c) \in L.\ s \overset{L,R}{\cong} f(c, s)$, and

(ii) $\forall (g, c) \in R.\ s \overset{L,R}{\cong} g(s, c)$.

We say that two terms $s, t \in \mathbb{T}(\Sigma)$ are *unit-context equivalent*, if $s \overset{L,R}{\cong} t$.

Since the sets $L$ and $R$ are always clear from the context, we write $\cong$ in place of $\overset{L,R}{\cong}$.

**Theorem 5.2 (Decidability of unit-context equivalence)** *Let $L, R \subseteq \Sigma \times \Sigma$ be finite sets of pairs of binary function symbols and constants. Then, for all terms $t, u \in \mathbb{T}(\Sigma)$, it is decidable whether $t \overset{L,R}{\cong} u$ holds.*

**Definition 5.3** [Left- and right-aligned pairs for unit elements] Given a GSOS language $G$, the sets $L$ and $R$ of pairs of binary function symbols and constants are the largest sets satisfying the following constraints.

**1.** For each $(f, c) \in L$, the following conditions hold:

**a.** For each action $a \in \mathcal{L}$, there exists at least one deduction rule of the form

$$\frac{\Phi \cup \{x_1 \xrightarrow{a} y_1\}}{f(x_0, x_1) \xrightarrow{a} t'},$$

where

**i.** $\models_G x_1 \xrightarrow{a} \Rightarrow \mathsf{hyps}(\Phi)[x_0 \mapsto c]$, and

**ii.** one of the following cases holds:

**A.** there are a premise $x_0 \xrightarrow{b} y \in \Phi$, for some $b \in \mathcal{L}$ and $y \in vars(t')$, and an axiom $c \xrightarrow{b} t$ such that $\sigma(t') \cong y_1$, where $\sigma$ is the substitution mapping $x_0$ to $c$, $y$ to $t$ and is the identity on all the other variables, or

**B.** $\sigma(t') \cong y_1$, where $\sigma$ is the substitution mapping $x_0$ to $c$ and is the identity on all the other variables.

**b.** For each $f$-defining deduction rule

$$\frac{\Phi}{f(x_0, x_1) \xrightarrow{a} t'}$$

one of the following cases holds:

**i.** $x_1 \xrightarrow{a} y_1 \in \Phi$ for some variable $y_1$ and

**A.** either there is a premise $x_0 \xrightarrow{b} y \in \Phi$, for some $b \in \mathcal{L}$ and variable $y \in vars(t')$, such that $c$ has a single axiom with label $b$—say, $c \xrightarrow{b} t$—and $\sigma(t') \cong y_1$, where $\sigma$ is the substitution mapping $x_0$ to $c$, $y$ to $t$ and is the identity on all the other variables,

**B.** or $\sigma(t') \cong y_1$, where $\sigma$ is the substitution mapping $x_0$ to $c$ and is the identity on all the other variables.

**ii.** $\models_G \mathsf{hyps}(\Phi)[x_0 \mapsto c] \Rightarrow \mathsf{False}$.

**2.** The definition of right-aligned pairs of operators and constant symbols—that is, those such that $(f, c) \in R$—is symmetric and is not repeated here.

For a function symbol $f$ and a constant $c$, we call $(f, c)$ *left aligned* (respectively, *right aligned*) if $(f, c) \in L$ (respectively, $(f, c) \in R$).

The following theorem states the correctness of the rule format defined above.

**Theorem 5.4** *Let $G$ be a GSOS language. Assume that $L$ and $R$ are the sets of left- and right-aligned function symbols according to Definition 5.3. For each $(f, c) \in L$, it holds that $f(c, x) \leftrightarrows x$. Symmetrically, for each $(f, c) \in R$, it holds that $f(x, c) \leftrightarrows x$.*

**Remark 5.5** The constraint that $c \xrightarrow{b} t$ be the only $c$-defining axiom with label $b$ in condition 1.b.i.A. of Definition 5.3 is necessary for the validity of Theorem 5.4. To see this, consider, for instance, the TSS over set of labels $\{a\}$ with constants $\mathbf{0}$, $\mathrm{RUN}_a$ and $c$, and the binary operator $\|_{\mathcal{L}}$ defined in Example 4.1. The rules for the constant $c$ are

$$\frac{}{c \xrightarrow{a} c} \qquad \frac{}{c \xrightarrow{a} \mathbf{0}} \ .$$

Observe that the sets $L = \{(\|_{\mathcal{L}}, c)\}$ and $R = \emptyset$ would satisfy the conditions in Definition 5.3 if the uniqueness requirement were dropped from condition 1.b.i.A. On the other hand, $c \,\|_{\mathcal{L}}\, \mathrm{RUN}_a$ is not bisimilar to $\mathrm{RUN}_a$ because

$$c \,\|_{\mathcal{L}}\, \mathrm{RUN}_a \xrightarrow{a} \mathbf{0} \,\|_{\mathcal{L}}\, \mathrm{RUN}_a \not\xrightarrow{a} \ ,$$

while $\mathrm{RUN}_a$ can only perform action $a$ forever. Therefore $c$ is *not* a left unit element for $\|_{\mathcal{L}}$. ⊠

The following result is a consequence of Theorems 2.6 and 5.2.

**Theorem 5.6** *For GSOS languages, the sets $L$ and $R$ can be effectively constructed.*

The format for left and right unit elements proposed above is incomparable to the one offered in [6]. Indeed, the latter allows for complex terms as source of the conclusions and in the premises, which the GSOS format forbids. On the other hand, in condition 1.a. above, the set of premises $\Phi$ may contain several tests on the argument variable $x_1$, which is forbidden by the purely syntactic format in [6]. A concrete, albeit admittedly inexpressive, example of a TSS exploiting this feature is discussed below.

**Example 5.7** Consider a TSS, over the set of labels $\{a, b\}$, with constants $\mathrm{RUN}_a$ and $\mathrm{RUN}_b$, and a binary function symbol $f$ defined by the rules below.

$$\frac{y \xrightarrow{a} y' \quad y \overset{b}{\not\to}}{f(x,y) \xrightarrow{a} y'} \qquad \frac{y \xrightarrow{b} y' \quad y \overset{a}{\not\to}}{f(x,y) \xrightarrow{b} y'}$$

The constants $\mathrm{RUN}_a$ and $\mathrm{RUN}_b$ are both left unit elements for $f$. Indeed, *every* closed term is a left unit element for $f$. This holds true because each closed term is bisimilar to one of the constants $\mathrm{RUN}_a$ and $\mathrm{RUN}_b$. Therefore, every process is either able to perform initially an $a$-transition or is able to perform initially a $b$-transition, but never both.

It is not hard to check that the sets $L = \{(f, \mathrm{RUN}_a), (f, \mathrm{RUN}_b)\}$ and $R = \emptyset$ satisfy the conditions in Definition 5.3. On the other hand, the format from [6] fails on this basic scenario since $y$ is tested twice in the rules for $f$. ⊠

All the examples from the literature mentioned in [6] can be handled by the rule format presented in Definition 5.3. By way of illustration, we limit ourselves to discussing just a single example addressed in [6].

**Example 5.8** [Synchronous Parallel Composition] Assume that $a$ is the only action in $\mathcal{L}$. Consider the constant $\mathrm{RUN}_a$ and the synchronous parallel composition operator $\|_{\mathcal{L}}$ from Example 4.1. For ease of reference, we recall that $\|_{\mathcal{L}}$ is specified by the rule

$$\frac{x \xrightarrow{a} x' \quad y \xrightarrow{a} y'}{x \,\|_{\mathcal{L}}\, y \xrightarrow{a} x' \,\|_{\mathcal{L}}\, y'} \quad (a \in \mathcal{L}) \ .$$

Take $L = R = \{(\|_{\mathcal{L}}, \mathrm{RUN}_a)\}$. These sets $L$ and $R$ meet the constraints in Definition 5.3. Let us discuss first the set $L$.

**1.a.** Consider the rule above. Since $(x \xrightarrow{a})[x \mapsto \mathrm{RUN}_a] = \mathsf{True}$, the entailment

$$\models_G y \xrightarrow{a} \Rightarrow (x \xrightarrow{a})[x \mapsto \mathrm{RUN}_a]$$

is trivially satisfied. Therefore condition 1.a.i. is met. Note, moreover, that $x \xrightarrow{a} x'$ is a premise of the rule above. Since we can pick the axiom

$$\frac{}{\mathrm{RUN}_a \xrightarrow{a} \mathrm{RUN}_a} \ ,$$

and the substitution $\sigma$ that maps $x$ and $x'$ to $\mathrm{RUN}_a$ and that is the identity function on all the other variables. Then, $\sigma(x' \|_{\mathcal{L}} y') \equiv \mathrm{RUN}_a \|_{\mathcal{L}} y' \cong y'$. Therefore condition 1.a.ii.A. is met.

**1.b.** Reasoning as above, we can easily check that rule above meets condition 1.b.i.A. in Definition 5.3.

A similar reasoning shows that $(\|_{\mathcal{L}}, \mathrm{RUN}_a)$ is also right aligned. ☒

## 6 Conclusions

In this paper we have provided a rule format ensuring that certain constants in a language act as left or right zero elements for a set of binary operators. The format for left and right zero elements presented in Section 3 follows the techniques developed by some of the authors in [6], where a format for left and right unit elements was offered, but the actual details are rather different.

The format makes use of the logic of initial transitions as proposed in [2] and is restricted to the so-called GSOS languages. It therefore does not include advanced features such as complex terms in the source of the conclusions of rules, like the one in [6] does for unit elements, but is still able to check relevant cases.

Following the design of the format for zero elements, we also provided an alternative rule format for left and right unit elements. Although this format is incomparable to the format from [6], it is still able to check all relevant cases from the literature and also some basic unit elements not addressed by the format from [6].

We believe that the formats we propose in this paper for GSOS languages are good candidates for mechanization in a tool-set for checking algebraic laws based on rule formats.

In [3], the full version of this paper, we also give a format for zero elements that is not restricted to GSOS languages, but follows the approach of [6] more closely, and apply it to a variety of examples from the literature. In this paper we have not included any material about the use of premises in deduction rules. In [3] we show that predicates can easily be dealt with.

## References

[1] Aceto, L., A. Birgisson, A. Ingolfsdottir, M. R. Mousavi and M. A. Reniers, *Rule formats for determinism and idempotence*, in: F. Arbab and M. Sirjani, editors, *Fundamentals of Software Engineering, Third IPM International Conference, FSEN 2009, Kish Island, Iran, April 15-17, 2009, Revised Selected Papers*, Lecture Notes in Computer Science **5961** (2010), pp. 146–161.

[2] Aceto, L., M. Cimini and A. Ingolfsdottir, *A bisimulation-based method for proving the validity of equations in GSOS languages*, in: *Proceedings of Structural Operational Semantics 2009, August 31, 2009, Bologna (Italy)*, Electronic Proceedings in Theoretical Computer Science **18**, 2010, pp. 1–16.

[3] Aceto, L., M. Cimini, A. Ingolfsdottir, M. Mousavi and M. A. Reniers, *On rule formats for zero and unit elements*, Technical Report CSR 10/03, TU/e (2010).

[4] Aceto, L., W. Fokkink and C. Verhoef, *Structural operational semantics*, in: *Handbook of Process Algebra* (1999), pp. 197–292.

[5] Aceto, L., A. Ingolfsdottir, M. Mousavi and M. A. Reniers, *Algebraic properties for free!*, Bulletin of the EATCS **99** (2009), pp. 81–104.

[6] Aceto, L., A. Ingolfsdottir, M. Mousavi and M. A. Reniers, *Rule formats for unit elements*, in: J. van Leeuwen, A. Muscholl, D. Peleg, J. Pokorný and B. Rumpe, editors, *SOFSEM 2010, 36th Conference on Current Trends in Theory and Practice of Computer Science, Špindleruv Mlýn, Czech Republic, January 23-29, 2010. Proceedings*, Lecture Notes in Computer Science **5901** (2010), pp. 141–152.

[7] Baeten, J., T. Basten and M. Reniers, "Process Algebra: Equational Theories of Communicating Processes," Cambridge Tracts in Theoretical Computer Science **50**, Cambridge University Press, 2009.

[8] Baeten, J. and C. Middelburg, "Process Algebra with Timing," Monographs in Theoretical Computer Science, An EATCS Series, Springer-Verlag, Berlin, 2002.

[9] Bergstra, J. and J. W. Klop, *Fixedpoint semantics in process algebra*, Technical Report IW 206/82, Center for Mathematics, Amsterdam, The Netherlands (1982).

[10] Bergstra, J. A. and C. A. Middelburg, *Preferential choice and coordination conditions*, J. Log. Algebr. Program. **70** (2007), pp. 172–200.

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

[12] Cranen, S., M. Mousavi and M. A. Reniers, *A rule format for associativity*, in: F. van Breugel and M. Chechik, editors, *Proceedings of the 19th International Conference on Concurrency Theory (CONCUR'08)*, Lecture Notes in Computer Science **5201** (2008), pp. 447–461.

[13] Hennessy, M. and R. Milner, *Algebraic laws for nondeterminism and concurrency*, J. ACM **32** (1985), pp. 137–161.

[14] Hoare, C. A. R., *Communicating sequential processes*, Commun. ACM **21** (1978), pp. 666–677.

[15] Milner, R., "Communication and concurrency," Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[16] Mousavi, M., M. Reniers and J. F. Groote, *A syntactic commutativity format for SOS*, Information Processing Letters **93** (2005), pp. 217–223.

[17] Mousavi, M. R., M. A. Reniers and J. F. Groote, *SOS formats and meta-theory: 20 years after*, Theor. Comput. Sci. **373** (2007), pp. 238–272.

[18] Park, D., *Concurrency and automata on infinite sequences*, in: *Proceedings of the 5th GI-Conference on Theoretical Computer Science* (1981), pp. 167–183.

[19] Plotkin, G. D., *A Structural Approach to Operational Semantics*, Technical Report DAIMI FN-19, University of Aarhus (1981).

[20] Plotkin, G. D., *A structural approach to operational semantics*, J. Log. Algebr. Program. **60-61** (2004), pp. 17–139.