Termination and confluence of term rewrite systems for GSOS process languages

Irek Ulidowski
Department of Mathematics and Computer Science
University of Leicester
http://www.mcs.le.ac.uk/~iulidowski/

Abstract

We consider the procedures for automatic derivation of axiom systems and term rewrite systems for process language in the GSOS format. We show that, for a decidable subclass of GSOS process languages, namely syntactically well-founded and linear GSOS process languages, the generated term rewrite systems are strongly normalising, confluent, and sound and complete for bisimulation modulo the associativity and commutativity of the choice operator on closed terms.

1 Introduction

Structural operational semantics (SOS) [Plo81, AFV01] is a method for assigning operational meaning to operators of process languages. The main feature of SOS are *transition rules*, or simply SOS rules, which describe how the behaviour of a composite process depends on the behaviour of its component processes. A *format* is a collection of general syntactic forms of transition rules. A process operator is in a format if all its SOS rules are in the format, and a process language, often abbreviated by PL, is in a format if all its operators are in the format. Many general formats have been proposed and a wealth of important results for PLs in these formats have been developed; these are catalogued in [AFV01]. In this paper we are interested in procedures for automatic derivation of sound and complete axioms systems for general PLs, and in procedures for turning these axiom systems into term rewrite systems.

We extend the work of Aceto, Bloom and Vaandrager [ABV94] and generalise the results of Bosscher [Bos94] on automatic generation term rewrite systems for PLs in the GSOS format [BIM95]. [ABV94] defines two procedures for generating sound and complete with respect to bisimulation axiom systems for arbitrary PLs in the GSOS format. The axiom systems created by the second of the two procedures can be turned into term rewrite systems by orienting the axioms from left to right. Bosscher shows that, for a decidable subclass of GSOS PLs, namely syntactically well-founded and linear GSOS PLs, the obtained term rewrite systems are strongly normalising (terminating). He also gives an indirect proof of confluence. However, the proofs rely on a somewhat strong assumption concerning the weight function w: it is assumed that $w(f) \geq 1$ for all operators f. In fact, the most straightforward weight functions for reasonably useful, syntactically well-founded GSOS PLs return values 0 for most of the operators.

Our contributions are as follows: We describe carefully, and make precise, the procedure for turning axiom systems for GSOS PLs into term rewrite systems, thus improving the presentation in [Bos94]. For syntactically well-founded and linear GSOS PLs we prove that the produced term rewrite systems are strongly normalising without any assumptions about the weight function. We use a novel technique based on dependency pairs [AG00]. Moreover, we show that the derived term rewrite systems are confluent and complete for bisimulation modulo the associativity and commutativity of the choice operator on closed terms. There are several examples which illustrate the application of the described axiomatisation and term rewriting procedures.

The paper is self-contained and the proofs are detailed. As a result the paper is regrettably one page too long.

2 Preliminaries

This section recalls the notions of processes, bisimulation and GSOS PLs. We assume a knowledge of basic definitions and results [Mil89, BIM95], but [BHR84, BW90, GV92] may also prove very useful.

Definition 2.1 A labelled transition system (or LTS, for short) is a structure $(\mathcal{P}, A, \rightarrow)$, where $\mathcal{P}$ is the set of processes, A is the set of actions and $\rightarrow \subseteq \mathcal{P} \times A \times \mathcal{P}$ is a *transition relation.*

We model concurrent systems by process terms (processes) which are the states in an LTS. Transitions between the states, defined by a transition relation, model the behaviour of systems. $\mathcal{P}$, the set of processes, is ranged over by $p, q, r, s, t, \ldots$. Act is a finite set of (visible and silent) actions and it is ranged over by a, b, c. We write $p \stackrel{a}{\rightarrow} q$ for $(p, a, q) \in \rightarrow$ and read it as process p performs a and in doing so becomes process q. Expressions of the form $p \stackrel{a}{\rightarrow} q$ are called *transitions*. We write $p \stackrel{a}{\rightarrow}$ when there is q such that $p \stackrel{a}{\rightarrow} q$, and $p \stackrel{a}{\nrightarrow}$ otherwise. We recall the definition of bisimulation [Mil89]:

Definition 2.2 Given $(\mathcal{P}, \mathsf{Act}, \rightarrow)$, a relation $R \subseteq \mathcal{P} \times \mathcal{P}$ is a bisimulation if, for all $(p, q) \in R$ and all $a \in \mathsf{Act}$, the following properties hold.

$$\begin{array}{lll} p \stackrel{a}{\rightarrow} p' & \text{implies} & \exists q'.(q \stackrel{a}{\rightarrow} q' \text{ and } p'Rq') \\ q \stackrel{a}{\rightarrow} q' & \text{implies} & \exists p'.(p \stackrel{a}{\rightarrow} p' \text{ and } p'Rq') \end{array}$$

$p \sim q$ if there exists a bisimulation R such that pRq.

2.1 GSOS Process Languages

Var is a countable set of variables ranged over by $X, X_i, Y, Y_i, \ldots$. Σ_n is a set of operators with arity n. A signature Σ is a collection of all Σ_n and it is ranged over by $f, g, \ldots$. The members of Σ_0 are called *constants*; $\mathbf{0} \in \Sigma_0$ is the deadlocked process constant. The set of *open terms* over Σ with variables in $V \subseteq \mathsf{Var}$, denoted by $\mathbb{T}(\Sigma, V)$, is ranged over by $t, t', \ldots$.

The set of *closed terms*, written as $\mathrm{T}(\Sigma)$, is ranged over by $p, q, u, v, \ldots$. In the setting of process languages these terms will often be called process terms. A Σ context with n holes $C[X_1, \ldots, X_n]$ is a member of $\mathbb{T}(\Sigma, \{X_1, \ldots, X_n\})$. If $t_1, \ldots, t_n$ are Σ terms, then $C[t_1, \ldots, t_n]$ is the term obtained by substituting t_i for X_i for $1 \leq i \leq n$. We will use bold italic font to abbreviate the notation for sequences (or vectors). For example, a sequence of process terms $p_1, \ldots, p_n$, for any $n \in \mathbb{N}$, will often be written as $\boldsymbol{p}$ when the length is understood from the context. A *substitution* is a mapping $\mathsf{Var} \rightarrow \mathrm{T}(\Sigma)$. Substitutions are ranged over by ρ, ρ' and σ; they extend to $\mathbb{T}(\Sigma, \mathsf{Var}) \rightarrow \mathrm{T}(\Sigma)$ mappings in a standard way.

Definition 2.3 A GSOS rule is an expression of the form

$$\frac{\{\ X_i \stackrel{a_{ij}}{\to} Y_{ij}\ \}_{i\in I, j\in J_i} \quad \{\ X_k \stackrel{b_{kl}}{\not\to}\ \}_{k\in K, l\in L_k}}{f(X_1,\ldots,X_n) \stackrel{\gamma}{\to} C[\boldsymbol{X},\boldsymbol{Y}],}$$

where $\boldsymbol{X}$ is the sequence $X_1,\ldots,X_n$ and $\boldsymbol{Y}$ is the sequence of all Y_{ij}, and all process variables in $\boldsymbol{X}$ and $\boldsymbol{Y}$ are distinct. Variables in $\boldsymbol{X}$ are the *arguments* of f. Moreover, I and K are subsets of $\{1,\ldots,n\}$ and all J_i and L_k, for $i\in I$ and $k\in K$, are finite subsets of $\mathbb{N}$, and $C[\boldsymbol{X},\boldsymbol{Y}]$ is a context.

Let r be the above GSOS rule. Then, f is the *operator* of r and the elements of $\boldsymbol{X}$ are the arguments of r. We write *rules*(f) for the set of GSOS rules for f. The expressions above the horizontal bar in r are called *premises*. The transition below the bar in r is the *conclusion*. $C[\boldsymbol{X},\boldsymbol{Y}]$ is the *target* of r. The rule has *negative* premises if it contains any $X \stackrel{a}{\not\to}$ premise. A transition in the premises is called a *positive* premise. The ith argument X_i is *active* in rule r if either $X_i \stackrel{a_{ij}}{\to} Y_{ij}$ or $X_k \stackrel{b_{kl}}{\not\to}$ is a premise. Consequently, the ith argument of $f(\boldsymbol{X})$ is active if it is active in some rule for f.

Multiple occurrences of process variables in rules are called *copies*. They are either *explicit* and *implicit* copies [Uli94]. Given a rule r as in Definition 2.3, explicit copies are the multiple occurrences of variables X_i in the premises, for $i\notin I$, and the multiple occurrences of Y_{ij} and X_i in $C[\boldsymbol{X},\boldsymbol{Y}]$ for $i\notin I$. The implicit copies are the multiple occurrences of X_i in the premises when $i\in I$, and the occurrences, not necessarily multiple, of variables X_i in $C[\boldsymbol{X},\boldsymbol{Y}]$ when $i\in I$.

Definition 2.4 A GSOS process language is a tuple (Σ, A, R), where Σ is a finite set of operators, $A\subseteq \mathsf{Act}$, R is a finite set of transition rules for operators in Σ such that all actions mentioned in the transition rules belong to A.

A GSOS operator is *smooth* if its rules have no implicit copies. A transition rule for a smooth GSOS operator is *discarding* if no argument that appears in a negative premise of the rule appears also in the target. A smooth operator is discarding if all its rules are discarding. A GSOS operator f is *distinctive* if, for each argument i, the argument is active either in all transition rules for f or in none of them, and also, for each pair of different rules for f, there is an argument for which both rules have the same premise but with a different action.

Given a GSOS PL $G=(\Sigma,A,R)$, we can associate a unique transition relation $\to$ with G [BIM95]. Then, we construct $(\mathrm{T}(\Sigma), A, \to)$, the LTS for G. Bisimulation is then defined as in Definition 2.2, and we have a powerful result stating that bisimulation is a congruence for all GSOS PLs [BIM95].

Definition 2.5 [ABV94] An GSOS PL H is a *disjoint extension* of a GSOS PL G, written as $G\leq H$, if the signature and rules of H include those of G, and H introduces no new rules for operators in G.

3 GSOS Axiom Systems

In this section we introduce the second axiomatisation procedure from [ABV94]. The input to the procedure is an arbitrary GSOS PL G with signature Σ_G. The output of the procedure is

- a GSOS PL H such that $G \leq H$ and $\mathsf{FINTREE} \leq H$, where $\mathsf{FINTREE}$ consists of $\mathbf{0}$, unary prefixing operators $a.\cdot$ for every $a \in \mathsf{Act}$ and the binary choice operator $+$, all as in [Mil89]; and

- a set of axioms E_H over Σ_H which are sound for bisimulation, namely if p is provably equal to q with respect to axioms E_H, written $(E_H \vdash)p = q$, then $p \sim q$ over the LTS for H, written $p \sim_H q$ or simply $p \sim q$ if H clear from the context.

The set E_H contains the following axioms for $\mathsf{FINTREE}$: $X + \mathbf{0} = X$, $X + X = X$, $X + Y = Y + X$ and $(X + Y) + Z = X + (Y + Z)$. They are sound and complete for bisimulation over $\mathsf{FINTREE}$ [Mil89], namely $E_{\mathsf{FINTREE}} \vdash p = q$ if and only if $p \sim_{\mathsf{FINTREE}} q$.

The operators in $H \setminus \mathsf{FINTREE}$ are of the three types: (a) operators which are not both smooth and discarding, (b) smooth, discarding but not distinctive operators, and (c) smooth, discarding and distinctive operators. The axiomatisation procedure generates certain types of axioms depending on the type of operators. Here, we only list the types of generated axioms; the precise details and many illuminating examples can be found in [ABV94].

Definition 3.1 Let $f \in \Sigma_H$ be an n-ary GSOS operator which is not both both smooth and discarding, and let $A \in E_H$ be an axiom of the form

$$f(\boldsymbol{X}) = f^c(\boldsymbol{Y}),$$

where $\boldsymbol{X}$ is a vector of n distinct variables and $\boldsymbol{Y}$ is a vector of m variables from $\boldsymbol{X}$ with $m > n$, and $f^c \in \Sigma_H$ with arity m is either smooth, discarding but not distinctive operator or smooth, discarding and distinctive operator. A is called the *copying* axiom for f.

There are no other axioms for the operators of H of type (a) above. Note, that each such operator of H belongs also to G, and the procedure extends G with the relevant operators f^c for each $f \in G$ which is not both smooth and distinctive.

Each operator of type (b) in H, namely smooth, discarding but not distinctive operator, has an axiom which links it to its auxiliary operators which have been added to G by the axiomatisation procedure.

Definition 3.2 Let $f \in \Sigma_H$ be an n-ary smooth, discarding but not distinctive GSOS operator, and let $A \in E_H$ be an axiom of the form

$$f(\boldsymbol{X}) = \sum_{i \in aux(f)} f_i(\boldsymbol{X}),$$

where $\boldsymbol{X}$ is a vector of n distinct variables, $aux(f) \subset \mathbb{N}$ and $|aux(f)| \leq |rules(f)|$, and all f_is are smooth, discarding and distinctive operators. A is called the *auxiliary* axiom for f.

Example 3.3 Consider $\mathsf{FINTREE}$ extended with the CCS parallel composition operator $\|$ [Mil89]. The operator is smooth but not distinctive. The auxiliary operators required by Definition 3.2 are the left-merge, written as $\lfloor\!\lfloor$, the right-merge, written as $\rfloor\!\rfloor$, and the communication merge, written as $|$, operators [BW90, ABV94]. The operators are defined by the following GSOS rules.

$$\frac{X \xrightarrow{a} X'}{X \lfloor\!\lfloor Y \xrightarrow{a} X' \| Y} \qquad \frac{Y \xrightarrow{a} Y'}{X \rfloor\!\rfloor Y \xrightarrow{a} X \| Y'} \qquad \frac{X \xrightarrow{a} X' \quad Y \xrightarrow{\bar{a}} Y'}{X \mid Y \xrightarrow{\tau} X' \| Y'}$$

The auxiliary axiom below appeared in [BW90]; it is also an instance of the distinctifying law [ABV94]:

$$X \| Y \;=\; X \mid Y \;+\; X \lfloor\!\lfloor Y \;+\; X \rfloor\!\rfloor Y$$

Finally, each operator of type (c), namely smooth, discarding and distinctive operator, has several types of axioms generated by the axiomatisation procedure. They are the well known distributivity, action and inaction axioms, as well as the less common peeling axioms.

Definition 3.4 Let $f \in \Sigma_H$ be an n-ary smooth, discarding and distinctive GSOS operator, and let $A \in E_H$ be an axiom. If A has the form

$$f(P_1, \ldots, b_k.X_k + Y_k, \ldots, P_n) = f(P_1, \ldots, Y_k, \ldots, P_n),$$

where there is a GSOS rule for f, as in Definition 2.3 where $I \cap K = \emptyset$, with negative premises $\{X_k \stackrel{a_{kj}}{\nrightarrow}\}$ such that $b_k \notin \{a_{kj}\}$, and each P_i is either $a_i.X_i$, for $i \in I$, or X_i for $i \neq k$ and $i \in K$, then A is called a *peeling* axiom. If A has the form

$$f(X_1, \ldots, X_i + Y_i, \ldots, X_n) = f(X_1, \ldots, X_i, \ldots, X_n) \ + \ f(X_1, \ldots, Y_i, \ldots, X_n),$$

where each SOS rule for f has a positive premise for argument i, then A is called a *distributivity* axiom. If A has the form

$$f(\boldsymbol{P}) = a.C[\boldsymbol{X}],$$

where there is an SOS rule for f of the form as in Definition 2.3, and each P_i is $a_i.X_i$ when $i \in I$, or is $\mathbf{0}$ when $i \in K$ and $|L_k| > 0$, and X_i otherwise, then A is called an *action* axiom. If A has the form

$$f(\boldsymbol{P}) = \mathbf{0},$$

where each P_i is either $\mathbf{0}$, $a_i.X_i$ or $c_i.X_i + Y_i$ and for each rule for f there is i such that either (1) $i \in I$ and P_i is $\mathbf{0}$ or P_i is $a_i.X_i$ and the rule has no premise $X_i \stackrel{a_i}{\rightarrow} X_i'$, or (2) $i \in K$ and P_i is $c_i.X_i + Y_i$ and the rule has a premise $X_i \stackrel{c_i}{\nrightarrow}$, then A is called an *inaction* axiom.

Example 3.5 We illustrate how the second procedure generates axioms for arbitrary GSOS operators by considering the priority operator θ [BBK86]. For a given partial order $>$ on actions process $\theta(P)$ is a restriction of P such that, in any state of P, action a can happen only if no action b with higher priority, $b > a$, is possible. If $B_a = \{b \mid b > a\}$, then the operator is defined in a natural fashion by the first of the following GSOS rules, one for each action a:

$$\frac{X \stackrel{a}{\rightarrow} X' \quad \{X \stackrel{b}{\nrightarrow}\}_{b \in B_a}}{\theta(X) \stackrel{a}{\rightarrow} \theta(X')} \qquad\qquad \frac{X \stackrel{a}{\rightarrow} X' \quad \{Y \stackrel{b}{\nrightarrow}\}_{b \in B_a}}{X \triangle Y \stackrel{a}{\rightarrow} \theta(X')}$$

The second procedure produces the axioms for θ that use FINTREE operators. Since a typical rule for θ may have several copies of the argument X in the premises the operator is not smooth. Hence, Definition 3.1 requires an auxiliary binary operator for θ. The operator is denoted $\triangle$ and is defined above. The axioms for θ consist of the copying axiom for θ (given below), and the axioms for $\triangle$ operator. $\triangle$ is smooth, distinctive and discarding operator, and we have peeling axioms for the second argument, distributivity axioms for the first argument, action axioms for the first argument, and inaction axioms for both arguments. Together with the copying axiom for θ we have [ABV94]:

$$\begin{array}{rcll}
\theta(X) & = & X \triangle X & \\
a.X \triangle (b.Y + Z) & = & a.X \triangle Z & \quad \text{if } \neg(b > a) \\
(X+Y) \triangle Z & = & X \triangle Z + Y \triangle Z & \\
a.X \triangle \mathbf{0} & = & a.\theta(X) & \\
\mathbf{0} \triangle X & = & \mathbf{0} & \\
a.X \triangle (b.Y + Z) & = & \mathbf{0} & \quad \text{if } b > a
\end{array}$$

Note, that there is no distributivity axiom for the second argument of $\triangle$ as no rule for $\triangle$ has the positive premise for the second argument.

Result 3.6 Let G be a GSOS PL, and let H and E_H be the GSOS PL and the axiom system respectively that are produced by the second procedure in [ABV94]. Then, E_H is head normalising and sound for bisimulation, namely if $E_H \vdash p = q$, then $p \sim_H q$.

4 Term Rewriting

We recall the basic notions of term rewriting [Klo92]. A Term Rewriting System $\mathcal{R}$ is a pair (Σ, R) where Σ is a signature and R is a set of *reduction rules* or *rewrite* rules. We associate a countably infinite set of variable $V \subseteq \mathsf{Vis}$ with each TRS. A reduction rule is a pair of terms (t, s) over $\mathbb{T}(\Sigma, V)$ and it is written as $t \to s$. Two conditions are imposed on the terms of reduction pairs: t is not a variable, and the variables of s are also variables of t, namely $var(s) \subseteq var(t)$. Often a reduction rule has a name, for example r, and we write $r : t \to s$.

A reduction rule $r : t \to s$ can be seen as a prescription for deriving *rewrites* $\sigma t \to \sigma s$ for all substitutions σ, where a rewrite is a closed instance of a reduction rule. The left-hand side σt is called a *redex*, more precisely r-redex. The right-hand side σs is called a *contractum*. A σt redex may be replaced by its contractum σs in an arbitrary context $C[\ ]$ giving rise to a *reduction step* (one-step rewriting): $C[\sigma t] \to_r C[\sigma s]$. We call $\to_r$ the *one-step reduction relation* generated by r. The one-step reduction relation of a TRS $\mathcal{R}$, denoted by $\to_{\mathcal{R}}$ or simply by $\to$, is defined as the union of $\to_r$ for all $r \in \mathrm{R}$. Let R be a set of rewrites. The closure of R under closed contexts is denoted by $\to_R$. The transitive and reflexive closure of $\to$ ($\to_R$) is called *reduction* (*R-reduction*) and is written as $\twoheadrightarrow$ ($\twoheadrightarrow_R$). If $t \twoheadrightarrow s$ ($t \twoheadrightarrow_R s$), then s is called a *reduct* (an *R-reduct*) of t. A reduction of term $f(t_1, \ldots, t_n)$ is *internal* if it occurs solely in the subterms $t_1, \ldots, t_n$ leaving the head operator f unaffected.

When no reduction step is possible from a term t, we say that t is a *normal from*. This happens when t has no redex occurrences. A term is called *weakly normalising* if is can be reduced to a normal form; t is *strongly normalising* if it has no infinite reductions; and t is called *confluent* if any two reducts of t are *convergent* (or *joinable*), namely have a common reduct. A TRS is weakly normalising, strongly normalising and confluent if all its terms have these relevant properties.

The notions that are very useful in proving confluence are *overlap* and *critical pair*. Two reduction rules $r_0 : l \to r$ and $r_1 : l' \to r'$ overlap if and only if there is a non-variable subterm of l that can be matched with an r_1-redex (or vice versa). More precisely, there is some context $D[\ ]$ and a non-variable term s such that $l \equiv D[s]$ and $\sigma s \equiv \rho l'$ for some substitutions σ and ρ. Next, consider a pair of overlapping reduction rules $r_0 : l \to r$ and $r_1 : l' \to r'$. We shall assume that σ and ρ are such that $\sigma s \equiv \rho l'$ is a most general common instance of s and l', and that σ is minimal. The pair of one-step reducts of the outer redex $\sigma l \equiv \sigma D[\rho l']$ that arises from this overlap, $(\sigma D[\rho r'], \sigma r)$, is called a critical pair.

In order to prove confluence we will use the result due Knuth and Bendix [Klo92] that states that if a TRS is strongly normalising, then it is confluent if and only if all its critical pairs are convergent.

4.1 Rewriting modulo the associativity and commutativity of $+$

The application of term rewriting in concurrency is somewhat complicated by the need to preserve the commutativity and associativity of the nondeterministic choice operator $+$. These properties of $+$ are represented by the equations e_1 and e_2 below.

$$\begin{array}{llcl} e_1: & X+Y & = & Y+X \\ e_2: & X+(Y+Z) & = & (X+Y)+Z \end{array}$$

The equations cannot be oriented without losing the normalising property. For example, if we turn e_1 into $X+Y \to Y+X$, then $t+s \to s+t \to t+s \to \cdots$. Therefore, we shall use term rewriting modulo the commutativity and the associativity of $+$ in this paper. We denote the axioms e_1 and e_2 by AC and the equivalence class of terms t under AC by $[t]_{AC}$. For terms t and s such that $t \in [t']_{AC}$ and $s \in [s']_{AC}$, and for some terms t' and s', if $t' \to s'$, then we shall write $[t]_{AC} \to_{AC} [s]_{AC}$. Similarly, we define $[t]_{AC} \twoheadrightarrow_{AC} [s]_{AC}$ and the internal reductions of $[t]_{AC} \twoheadrightarrow_{AC} [s]_{AC}$. Henceforth, we shall drop all subscripts AC.

Definition 4.1 Let $G = (\Sigma, A, R)$ be a GSOS process language, and let $\mathcal{T} = (\Sigma, \mathrm{R})$ be a TRS for the rewrite set P. A rewrite $t \to s$ of P, where t and s are closed Σ terms, is *sound for bisimulation* if $t \sim s$. A rewrite rule $\mathrm{R} \ni r_0 : l \to r$ is sound for bisimulation if every r_0-rewrite is sound for bisimulation. $\mathcal{T}$ is sound for bisimulation if all its rewrite rules are. The set P is *complete for bisimulation* if, for all closed Σ terms t and s, whenever $t \sim s$, then t P-reduces to normal form t', s P-reduces to normal form s' and $[t'] = [s']$. $\mathcal{T}$ is complete for strong bisimulation, if its rewrite set P is.

Example 4.2 Let $(\Sigma_{\mathsf{FINTREE}}, \mathrm{R})$ be a TRS with the following set R of reduction rules.

$$\begin{array}{llcl} +_{nil}: & X+\mathbf{0} & \to & X \\ +_{ice}: & X+X & \to & X \end{array}$$

Term $a.X+(a.X+\mathbf{0})$ reduces to $a.X$ as follows: $a.X+(a.X+\mathbf{0}) \to_{+_{nil}} a.X+a.X \to_{+_{ice}} a.X$. There is another reduction modulo AC to $a.X$: $a.X+(a.X+\mathbf{0}) = (a.X+a.X)+\mathbf{0} \to_{+_{ice}} a.X+\mathbf{0} \to_{+_{nil}} a.X$. Hence, $[a.X+(a.X+\mathbf{0})] \twoheadrightarrow [a.X]$.

$(\Sigma_{\mathsf{FINTREE}}, \mathrm{R})$ is strongly normalising. Interpret $\mathbf{0}$, $a.X$ and $X+Y$ as polynomials 2, $2X$ and $X+Y$ to obtain polynomial termination modulo AC. Our TRS is also confluent modulo AC. Reduction rules $+_{nil}$ and $+_{ice}$ have a simple overlap which replaces X with $\mathbf{0}$. Now, we have $\mathbf{0}+\mathbf{0} \to_{+_{nil}} \mathbf{0}$ and $\mathbf{0}+\mathbf{0} \to_{+_{ice}} \mathbf{0}$. Hence, there is only one critical pair $([\mathbf{0}],[\mathbf{0}])$, and it is joinable.

$(\Sigma_{\mathsf{FINTREE}}, \mathrm{R})$ is sound and complete for bisimulation modulo AC [BW90].

5 Rewrite rules for GSOS process operators

Each GSOS PL produced by the second procedure contains FINTREE. The rewrite rules for FINTREE have been given in Example 4.2.

For each of the non-FINTREE operators of the GSOS PL that the second procedure produces, and for each of the types of axioms for these operators we obtain rewrite rules by orienting the axioms from left to right. The rewrite rules are named after the axioms they originate form. We list these rewrite rules below, and refer the reader to Section 3 for the definitions of the types of operators and the types of their arguments that are used in the rewrite rules below.

$$
\begin{array}{rlll}
f_{copy}: & f(\boldsymbol{X}) & \rightarrow & f^c(\boldsymbol{Y}) \\
f_{aux}: & f(\boldsymbol{X}) & \rightarrow & \sum_{i \in aux(f)} f_i(\boldsymbol{X}) \\
f_{pel}: & f(P_1, \ldots, b_i.X_i + Y_i, \ldots, P_n) & \rightarrow & f(P_1, \ldots, Y_i, \ldots, P_n), \\
f_{dis}: & f(X_1, \ldots, X_i + Y_i, \ldots, X_n) & \rightarrow & f(X_1, \ldots, X_i, \ldots, X_n) \ + \ f(X_1, \ldots, Y_i, \ldots, X_n) \\
f_{act}: & f(\boldsymbol{P}) & \rightarrow & a.C[\boldsymbol{X}] \\
f_{nil}: & f(\boldsymbol{P}) & \rightarrow & \mathbf{0}
\end{array}
$$

We shall need two further types of rewrite rules to achieve confluence. We notice that f_{pel} and f_{nil} may overlap with $+_{nil}$ on the term of the form $b_i.X_i + \mathbf{0}$. Consider a f_{pel} redex $f(P_1, \ldots, b_i.X_i + \mathbf{0}, \ldots, P_n)$; it can be reduced either by f_{pel} or by $+_{nil}$ as follows:

$$
\begin{array}{lll}
f(P_1, \ldots, b_i.X_i + \mathbf{0}, \ldots, P_n) & \rightarrow_{f_{pel}} & f(P_1, \ldots, \mathbf{0}, \ldots, P_n) \\
f(P_1, \ldots, b_i.X_i + \mathbf{0}, \ldots, P_n) & \rightarrow_{f_{nil}} & f(P_1, \ldots, b_i.X_i, \ldots, P_n)
\end{array}
$$

The right-hand sides cannot be joined unless we have an *extra peeling* rewrite rule below, which is sound for bisimulation. Similarly, we have an *extra inaction* rewrite rule below for every inaction rule with the $i$the argument $b_i.X_i + Y_i$.

$$
\begin{array}{rlll}
f_{epel}: & f(P_1, \ldots, b_i.X_i, \ldots, P_n) & \rightarrow & f(P_1, \ldots, \mathbf{0}, \ldots, P_n) \\
f_{enil}: & f(P_1, \ldots, b_i.X_i, \ldots, P_n) & \rightarrow & \mathbf{0}
\end{array}
$$

Example 5.1 The TRS for FINTREE extended with θ contains the usual rewrite rules for FINTREE, the rewrite rules obtained by orienting from left to right the axioms from Example 3.5, and the extra peeling and inaction rules for $\triangle$:

$$
\begin{array}{rllll}
\triangle_{epel}: & a.X \triangle b.Y & \rightarrow & a.X \triangle \mathbf{0} & \text{if } \neg(b > a) \\
\triangle_{enil}: & a.X \triangle b.Y & \rightarrow & \mathbf{0} & \text{if } b > a
\end{array}
$$

Definition 5.2 Let G be a GSOS PL, and let H and E_H be the GSOS PL and the axiom system respectively that are produced for G by the second procedure in [ABV94]. A TRS (Σ_H, R_H), where R_H consists of $+_{nil}$ and $+_{ice}$, all the rewrite rules for the non-FINTREE operators of H obtained by orienting the axioms (in E_H) for the operators from left to right, as well as all the extra peeling and extra inaction rewrite rules, is called the TRS *associated* with E_H, or the TRS for G.

Theorem 5.3 Let G be a GSOS PL. Then, $\mathcal{T}$, the TRS for G, is head normalising and sound for bisimulation.

6 Termination

In this section we argue that, for a decidable subclass of GSOS PLs, namely syntactically well-founded and linear GSOS PLs, the TRSs for these PLs are strongly normalising (terminating) modulo AC on closed terms.

Let G be a GSOS PL. A term $p \in \mathrm{T}(\Sigma_G)$ is *well-founded* if there exists no infinite sequence $p_0, a_0, p_1, a_1, \ldots$ with $p \equiv p_0$ and $p_i \xrightarrow{a_i} p_{i+1}$ for all $i \geq 1$. G is well-founded if all its terms are well-founded. Well-foundedness of GSOS PLs is not decidable, but syntactical well-foundedness is decidable [ABV94]. Moreover, if a GSOS PL is linear as well as syntactically well-founded (as defined below), then its is well-founded [ABV94]. A GSOS transition rule as in Definition (2.3) is *linear* if each variable occurs at most once in the target and, for each active argument i, X_i does not occur in the target and at most one of the variables Y_{ij} does. A GSOS operator is linear if all its transition rules are linear. A GSOS PL is linear if all its operators are linear. An GSOS PL G is *syntactically well-founded* if and only if there exists a function $w : \Sigma_G \to \mathbb{N}$ such that, for each rule r of G with the operator f and target $C[\boldsymbol{X}, \boldsymbol{Y}]$, the following conditions hold,

- If r has no premises, then $w(f) > W(C[\boldsymbol{X}, \boldsymbol{Y}])$; and $w(f) \geq W(C[\boldsymbol{X}, \boldsymbol{Y}])$ otherwise,

where $W : \mathbb{T}(\Sigma_G) \to \mathbb{N}$ is given by $W(X) \stackrel{def}{=} 0$ and $W(f(t_1, \ldots, t_n)) \stackrel{def}{=} w(f) + W(t_1) + \cdots + W(t_n)$. **Remark**: Since prefixing is defined by rules with no premises, $a.X \xrightarrow{a} X$, we must have $w(a.) \geq 1$.

Let G be a syntactically well-founded and linear GSOS PL. Then, H, the GSOS PL produced for G by the second procedure in [ABV94], is also syntactically well-founded and linear. In particular, the weight function w over Σ_H is the extension of w over Σ_G such that $w(f^c) = w(f)$ for each pair f^c, f as in Definition 3.1, and $w(f_i) = w(f)$ for each pair of f_i, f as in Definition 3.2.

We shall also use two other weight functions.

Definition 6.1 Functions W', $e : \mathbb{T}(\Sigma_H) \to \mathbb{N}$ are defined as follows:

- $W'(X) \stackrel{def}{=} 0$, $W'(t_1{+}t_2) \stackrel{def}{=} max(W'(t_1), W'(t_2))$ and $W'(t_1 \rhd t_2) \stackrel{def}{=} max(W'(t_1), W'(t_2))$,
- $W'(f(t_1, \ldots, t_n)) \stackrel{def}{=} 1 + W'(t_1) + \cdots + W'(t_n)$ if f is any prefixing operator.
- $W'(f(t_1, \ldots, t_n)) \stackrel{def}{=} w(f) + W'(t_1) + \cdots + W'(t_n)$ if f is not $+$, $\rhd$ and any prefixing operator.

For any term t, $e(t)$ is 1 if t contains $+$, or a non-distinctive operator or a distinctive operator that has no active arguments (excluding $\mathbf{0}$), and 0 otherwise. Note, that constants and prefixing operators are distinctive operators that have no active arguments.

Definition 6.2 An ordering $\sqsupset$ over Σ_H terms is defined as follows: $t \sqsupset s$ if and only if (1) $e(t) > e(s)$, or (2) $e(t) = e(s)$ and $W'(t) > W'(s)$. An ordering $\sqsupseteq$ is a union of $\sqsupset$ and $\{(t, s) \mid W'(t) = W'(s) \text{ and } e(t) = e(s)\}$.

It is clear that $\sqsupset$ is a well-founded partial order closed under substitution. Clearly, $\sqsupseteq$ is reflexive and transitive and closed under substitution. Moreover, $\sqsupset$ is strictly monotonic and $\sqsupseteq$ is weakly monotonic.

Theorem 6.3 Let G be a syntactically well-founded and linear GSOS PL, and let H and E_H be the GSOS PL and the axiom system respectively that are produced for G by the second procedure in [ABV94]. Assume that $\mathcal{T}$ is the TRS for G. Then, $\mathcal{T}$ is strongly normalising for closed terms modulo associativity and commutativity of $+$.

Proof. Assume that $\mathcal{T} = (\Sigma_H, \mathrm{R})$ is the TRS for G. We employ the novel *dependency pair* and *dependency graph* techniques due to Arts and Giesl [AG00]. A proof of termination of PRSes (*Priority Rewrite Systems*) for syntactically well-founded and linear OSOS process languages is given in [Uli02]. It also uses the dependency pair and dependency graph techniques.

An operator $f \in \Sigma_H$ is a *defined symbol* if it appears as the head operator of the left-hand side of some rewrite rule in R. An $f \in \Sigma_H$ is a *constructor* if it is not a defined symbol. If $f(s_1, \ldots, s_n) \to C[g(t_1, \ldots, t_m)]$ is a rewrite rule of R with g a defined symbol, then

$$\langle f(s_1, \ldots, s_n), g(t_1, \ldots, t_m) \rangle$$

is called a *dependency pair* of $\mathcal{T}$. A sequence of dependency pairs $\langle s_1, t_1 \rangle, \langle s_2, t_2 \rangle, \ldots$ is and $\mathcal{T}$*-chain* if there exists a substitution σ such that $\sigma t_j \to \sigma s_{j+1}$ holds for every two consecutive pairs $\langle s_j, t_j \rangle$ and $\langle s_{j+1}, t_{j+1} \rangle$ in the sequence. When $\mathcal{T}$ is clear from the context we often write chain instead of $\mathcal{T}$-chain. The *dependency graph* of $\mathcal{T}$ is the directed graph whose nodes are the dependency pairs and there is an arc from $\langle s, t \rangle$ to $\langle v, w \rangle$ if $\langle s, t \rangle \langle v, w \rangle$ is a T-chain.

Result 6.4 [AG00] A TRS $\mathcal{T}$ is strongly normalising if and only if there exists a well-founded weakly monotonic quasi-ordering $\sqsupseteq$, where both $\sqsupseteq$ and its strongly monotonic version $\sqsupset$ are closed under substitution, such that

1. $l \sqsupseteq r$ for all rewrite rules $l \to r$ of $\mathcal{T}$;
2. $s \sqsupseteq t$ for all dependency pairs $\langle s, t \rangle$ on a cycle of the dependency graph for $\mathcal{T}$; and
3. $s \sqsupset t$ for at least one dependency pair $\langle s, t \rangle$ on each cycle of the dependency graph for $\mathcal{T}$.

The required quasi-ordering $\sqsupseteq$ is defined in terms of the weight functions from Definition 6.1, which are in turn based on w and W functions for syntactically well-founded PLs. Unlike in [Bos94], our proof does not rely on the assumption that $w(f) \geq 1$ for all f. In fact, for most of the existing PLs, the simplest required function w is such that $w(f) = 0$ for most operators f. Our proof works with $w(f) \geq 0$.

We return to our proof. The only constructors in $\mathcal{T} = (\Sigma_H, \mathrm{R})$ are the prefixing operators $a.$, for all $a \in A$, the operator $\mathbf{0}$ and possibly other constants that have no defining rules. Next, we work out the dependency pairs for $\mathcal{T}$. There are no dependency pairs for $\mathsf{FINTREE}$. The following are all the possible dependency pairs modulo AC that arise from rewrite rules in R, where L is any nonempty subset of $aux(f)$ and $C[\boldsymbol{X}] \equiv D[g(\boldsymbol{Y})]$ for some context $D[\,]$:

$$\langle f(\boldsymbol{X}),\ f^c(\boldsymbol{Y}) \rangle \tag{1}$$
$$\langle f(\boldsymbol{X}),\ \sum_{i \in L} f_i(\boldsymbol{X}) \rangle \tag{2}$$
$$\langle f(P_1, \ldots, b_i.X_i + Y_i, \ldots, P_n),\ f(P_1, \ldots, Y_i, \ldots, P_n) \rangle, \tag{3}$$
$$\langle f(P_1, \ldots, b_i.X_i, \ldots, P_n),\ f(P_1, \ldots, \mathbf{0}, \ldots, P_n) \rangle, \tag{4}$$
$$\langle f(X_1, \ldots, X_i + Y_i, \ldots, X_n),\ f(X_1, \ldots, X_i, \ldots, X_n)\ +\ f(X_1, \ldots, Y_i, \ldots, X_n) \rangle \tag{5}$$
$$\langle f(X_1, \ldots, X_i + Y_i, \ldots, X_n),\ f(X_1, \ldots, X_i, \ldots, X_n) \rangle \tag{6}$$
$$\langle f(\boldsymbol{P}),\ C[\boldsymbol{X}] \rangle \tag{7}$$
$$\langle f(\boldsymbol{P}),\ g(\boldsymbol{Y}) \rangle \tag{8}$$
$$\langle f(P_1, \ldots, b_i.X_i + Y_i, \ldots, P_n),\ \mathbf{0} \rangle \tag{9}$$
$$\langle f(P_1, \ldots, b_i.X_i, \ldots, P_n),\ \mathbf{0} \rangle \tag{10}$$

With the above listed types of dependency pairs we construct the dependency graph for $\mathcal{T}$. We do not describe the whole graph; by Result 6.4 it is sufficient to identify only the cycles. There are

- cycles created by self-embedding dependency pairs (3), (6) and (8) when $g = f$, and
- cycles created by several pairs one of which is an instance of (7) or (8). For example, the shortest cycles are of the form $\langle f(\boldsymbol{X}),\ f_i(\boldsymbol{X})\rangle$, $\langle f_i(\boldsymbol{P}), f(\boldsymbol{Y})\rangle$, where the last pair is an instance of (8). There may be longer cycles that involve more than two operators, but they are made up solely from instances of (1), (2), (3), (6) and (8). The common property of all such cycles is that they contain an instance of (8).

Now, we show that the quasi-ordering $\sqsupseteq$ satisfies the three conditions of Result 6.4.

1. We show that $l \sqsupseteq r$ for all rewrite rules of $\mathcal{T}$ as in Section 5. We begin with rewrite rules for $\mathsf{FINTREE}$: $+_{nil} : X + \mathbf{0} \sqsupseteq X$ since $e(X + \mathbf{0}) \geq 1 > 0 = e(X)$; and $+_{ice} : X + X \sqsupseteq X$ since $e(X + X) = 1 > 0 = e(X)$. Next, we consider rewrite rules for operators $f \in \Sigma_H \setminus \Sigma_{\mathsf{FINTREE}}$. Assume that m is the number of active arguments of f.

   (a) f_{copy}. We have $f(\boldsymbol{X}) \sqsupseteq f^c(\boldsymbol{Y})$ since $e(f(\boldsymbol{X})) = 1 > 0 = e(f^c(\boldsymbol{Y}))$. $e(f^c(\boldsymbol{Y})) = 0$ as f^c is distinctive and it has active arguments.

   (b) f_{aux}. Since f is not distinctive we have $e(lhs) = 1 = e(rhs)$ as rhs contains $+$. Recall that $w(f) = w(f_i)$ for all $1 \leq i \leq l$. Thus, $W'(lhs) = w(f) = W'(rhs)$. Hence, $rhs = lhs$.

   (c) f_{pel}. $e(lhs) = 1 \geq e(rhs)$, and since $W'(b_i.X_i + Y_i) > W'(Y_i)$, we have $W'(lhs) > W'(rhs)$.

   (d) $f_{ds}(i) : f(\ldots, X_i + Y_i, \ldots) \to f(\ldots, X_i, \ldots) + f(\ldots, Y_i, \ldots)$. We easily verify that $rhs = lhs$.

   (e) $f_{act} : f(\boldsymbol{P}) \to a.C[\boldsymbol{X}]$. Clearly, $e(lhs) = 1 = e(rhs)$ since either prefixing occurs in $\boldsymbol{P}$ or f is distinctive operator with no active arguments. We have two cases for f. Assume that f has $m \geq 1$ active arguments. We have $W'(f(\boldsymbol{P})) \geq w(f) + m \geq W(C[\boldsymbol{X}]) + m$ since $w(f) \geq W(C[\boldsymbol{X}])$ by syntactical well-foundedness. Now, $m + W(C[\boldsymbol{X}]) \geq m + W'(C[\boldsymbol{X}]) \geq W'(a.C[\boldsymbol{X}])$.
   When f has no active arguments, and its action rewrite rule has the form $f(\boldsymbol{X}) \to a.C[\boldsymbol{X}]$, then $W'(f(\boldsymbol{P})) = W'(f(\boldsymbol{X})) = w(f) \geq 1 + W(C[\boldsymbol{X}])$ since $w(f) > W(C[\boldsymbol{X}])$. Hence, $1 + W(C[\boldsymbol{X}]) \geq 1 + W'(C[\boldsymbol{X}]) = W'(a.C[\boldsymbol{X}])$.

   (f) $f_{nil} : f(\boldsymbol{P}) \to \mathbf{0}$. There are three cases. Firstly, one of P_is involves prefixing; secondly, f is a constant with no defining rules; thirdly, f is not a constant with no defining rules and no P_is involves prefixing. In the last case one of P_is must be $\mathbf{0}$ [ABV94]. In the first two case $e(lhs) = 1 > 0 = e(rhs)$. In the last case $lhs \sqsupseteq rhs$.

2. $lhs \sqsupseteq rhs$ holds for dependency pairs (1), (2), (3), (6) and (8) by (a), (b), (c), (d) and (e) above.

3. Each cycle in our dependency graph contains at least one of the three types of dependency pairs: (3), (6) or (8). Since (7) is more general than (8) we show $lhs \sqsupset rhs$ for (3), (6) and (7).

   (a) $\langle f(P_1, \ldots, b_i.X_i + Y_i, \ldots, P_n),\ f(P_1, \ldots, Y_i, \ldots, P_n)\rangle$: clearly $e(lhs) = 1 \geq e(rhs)$, and and since $W'(b_i.X_i + Y_i) > W'(Y_i)$, we have $W'(lhs) > W'(rhs)$.

(b) (6): since $e(lhs) = 1$ and $e(rhs) = 0$, as f is distinctive operator with active arguments, we have $e(lhs) > e(rhs)$.

(c) $\langle f(\boldsymbol{P}),\ C[\boldsymbol{X}]\rangle$: We have $e(lhs) \geq e(rhs)$. Moreover, if $m \geq 1$, then $W'(f(\boldsymbol{P})) \geq w(f)+m \geq W(C[\boldsymbol{X}])+m \geq W'(C[\boldsymbol{X}])+m > W'(C[\boldsymbol{X}])$. If $m = 0$, namely f has no active arguments and its action rewrite rule has the form $f(\boldsymbol{X}) \rightarrow a.C[\boldsymbol{X}]$, then $W'(f(\boldsymbol{X})) = w(f) > W(C[\boldsymbol{X}])$, by syntactical well-foundedness, and $W(C[\boldsymbol{X}]) \geq W'(C[\boldsymbol{X}])$. Hence, $lhs \sqsupset rhs$. ■

7 Confluence and Completeness for Bisimulation

We show that if a TRS, as in the previous section, is strongly normalising, then it is also confluent. To prove this we shall use the classical result of Knuth and Bendix [Klo92] that states that if a TRS is strongly normalising, then it is is confluent if and only if all its critical pairs are convergent.

Theorem 7.1 Let G be a syntactically well-founded and linear GSOS PL, and let H and E_H be the GSOS PL and the axiom system respectively that are produced for G by the second procedure in [ABV94]. Assume that $\mathcal{T}$ is the TRS associated with E_H. Then, $\mathcal{T}$ is is confluent on closed terms over E_H.

Proof. We consider all possible critical pairs. There are no critical pairs in the FINTREE component.

1. The rule $X+\mathbf{0} \rightarrow X$ overlaps with a peeling rule f_{pel} when P_i is $b_i.X_i+Y_i$. The resulting critical pair is joinable using of the relevant extra peeling rule and $+_{nil}$.

2. The rule $X+\mathbf{0} \rightarrow X$ overlaps with rules f_{dis} for all distinctive operators f and for all their active arguments i. The resulting critical pairs are joinable since $f(\ldots,\mathbf{0},\ldots) \rightarrow_{f_{act}} \mathbf{0}$.

3. The rule $X+\mathbf{0} \rightarrow X$ overlaps with each f_{nil} rule when P_i is $b_i.X_i+Y_i$. The resulting critical pair is joinable using of the relevant extra inaction rule and $+_{nil}$.

4. The rule $X+X \rightarrow X$ overlaps with distributivity rules f_{dis} for all distinctive operators f and all their active arguments i. $(f(\ldots,X_i,\ldots), f(\ldots,X_i,\ldots)+f(\ldots,X_i,\ldots))$, the resulting critical pairs, are clearly joinable using $+_{ice}$.

5. Rules f_{pel} and f_{nil} overlap when P_i is $b_i.X_i+c_i.X_i'+Y_i$ and i is as in Definition 3.4. The resulting critical pair $\langle f(\ldots,c_i.X_i'+Y_i,\ldots),\mathbf{0}\rangle$ is joinable as $f(\ldots,c_i.X_i'+Y_i,\ldots) \rightarrow_{f_{nil}} \mathbf{0}$.

No other rewrite rules overlap. Note, that f_{pel} (f_{epel}) and f_{act} do not overlap due to the form of their arguments: f_{act} is applicable precisely when all the arguments that could have had the form $b_i.X_i+Y_i$ ($b_i.X_i$), and thus could have made f_{pel} (f_{epel}) applicable, are actually $\mathbf{0}$. See Definition 3.4. ■

Finally, we consider completeness for bisimulation.

Theorem 7.2 Let $\mathcal{T}$ be the TRS as in Theorem 7.1. Then, $\mathcal{T}$ is complete for bisimulation.

Proof. Since, $\mathcal{T}$ is strongly normalising it is well-defined. Since $\mathcal{T}$ is confluent each closed H term can be reduced to a unique normal form. As $\mathcal{T}$ is head normalising, and H is well-founded, we show by structural induction that each closed H term can be reduced to a unique FINTREE term in nf. Since $\mathcal{T}$ is sound for bisimulation, it is now sufficient to prove that the TRS for FINTREE is complete for bisimulation. This is done in Example 4.2. ■

8 Conclusion

We have considered a procedure for automatic derivation of term rewrite systems for process language in the GSOS format [ABV94, Bos94]. We have shown that, for a decidable subclass of GSOS process languages, namely syntactically well-founded and linear GSOS process languages, the generated term rewrite systems are strongly normalising, confluent, and sound and complete for bisimulation modulo AC on closed terms. We claim that a similar proof method, based on the dependency pairs technique, can be used to show weak normalisation and confluence for well-founded GSOS PLs.

References

[ABV94] L. Aceto, B. Bloom, and F.W. Vaandrager. Turning SOS rules into equations. *Information and Computation*, 111:1–52, 1994.

[AFV01] L. Aceto, W. Fokkink, and C. Verhoef. Structured operational semantics. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, pages 197–292. Elsevier Science, 2001.

[AG00] T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.

[BBK86] J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. Syntax and defining equations for an interrupt mechanism in process algebra. *Fundamenta Informaticae*, XI(2), 1986.

[BHR84] S.D. Brookes, C.A.R. Hoare, and W. Roscoe. A theory of communicating sequential processes. *Journal of the ACM*, 31:560–599, 1984.

[BIM95] B. Bloom, S. Istrail, and A.R. Meyer. Bisimulation can't be traced. *Journal of the ACM*, 42(1):232–268, 1995.

[Bos94] D.J.B. Bosscher. Term rewriting properties of SOS axiomatisations. In *Proceedings of TACS'94*, volume 1000 of *LNCS*, Berlin, 1994. Springer.

[BW90] J.C.M. Baeten and W.P. Weijland. *Process Algebra*, volume 18 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1990.

[GV92] J.F. Groote and F. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Information and Computation*, 100:202–260, 1992.

[Klo92] J.W. Klop. Term rewriting systems. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, pages 1–116. Oxford University Press, 1992.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[Plo81] G. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, 1981.

[Uli94] I. Ulidowski. *Local Testing and Implementable Concurrent Processes.* PhD thesis, Imperial College, University of London, 1994.

[Uli02] I. Ulidowski. Priority rewrite systems for OSOS process languages. A draft paper available at http://www.mcs.le.ac.uk/~iulidowski/PRS.ps, 2002.