Generating Priority Rewrite Systems for OSOS Process Languages ⋆

Irek Ulidowski [a] and Shoji Yuen [b]

[a] *Department of Computer Science, University of Leicester, University Road, Leicester, LE1 7RH, United Kingdom*

[b] *Information Engineering Department, Nagoya University, Furo-cho, Chikuka-ku, Nagoya 464-8601, Japan*

Abstract

We propose an algorithm for generating a Priority Rewrite System (PRS) for an arbitrary process language in the OSOS format such that rewriting of process terms is sound for bisimulation and head normalising. The algorithm is inspired by a procedure which was developed by Aceto, Bloom and Vaandrager and presented in *Turning SOS rules into equations* [2].

For a subclass of OSOS process languages representing finite behaviours the PRSs that are generated by our algorithm are strongly normalising (terminating) and confluent, where termination is proved using the dependency pair and dependency graph techniques. Additionally, such PRSs are complete for bisimulation on closed process terms modulo associativity and commutativity of the choice operator of CCS. We illustrate the usefulness of our results, and the benefits of rewriting with priorities in general, with several examples.

1 Introduction

Structural Operational Semantics (SOS) [31,3] is a method for assigning operational meaning to operators of process languages. The main components 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 general syntactic form of transition rules is called a *format*. 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 and specification and verification methods for PLs in these formats have been developed [3].

⋆ An extended abstract of this work appeared at CONCUR 2003 as [39].

The motivation and rationale for working with general PLs (via their formats) rather than with specific PLs such as, for example, CCS [27], CSP [20] and ACP [10], is that one can define and use new application-specific operators and features on top of the standard PLs [11,12]. In order to realise the potential of general PLs software tools need to be developed. Such tools would accept general PLs as input languages and perform tasks such as simulation, model checking and equivalence checking, refinement and testing. Several such tools already exist. For example, we can use *Process Algebra Compiler* [36] to change the input PL to the *Concurrency Workbench of New Century* [16]. The Process Algebra Compiler can accept any general PL in the *positive GSOS* format [13] and it produces a "front-end" to the Concurrency Workbench for that PL.

Alternatively, we can utilise the existing term rewriting and theorem prover software tools to analyse properties of processes of general PLs. To this end several procedures for automatic derivation of axiom systems and term rewriting systems for PLs in several formats were proposed [2,1,14,38,8]. The present paper continues this research, particularly on the generation of term rewriting systems for bisimulation originated by Aceto, Bloom and Vaandrager [2] and Bosscher [14], and extends and generalises it further. We propose a new procedure for deriving *Priority Rewrite Systems* for bisimulation. Having considered many examples of operators we believe that our work delivers the following improvements: (a) priority rewrite rules are no more complicated and are sometimes simpler than the rewrite rules produced from the axioms as in [2,1,14], (b) they employ no more than and sometimes fewer auxiliary operators (see Remark 5.2), and (c) the priority order that we use increases the effectiveness of term rewriting by reducing the number of critical pairs and thus reducing the nondeterminism inherent in rewriting (see Section 7). We work with *Ordered* SOS PLs [40], or OSOS PLs for short, instead of the GSOS PLs [13] which have the same expressiveness [40]. The proposed procedure generates term rewriting systems with a priority order on rewrite rules instead of axiom systems or ordinary term rewriting systems as in [2,14]. We illustrate this with an example. Consider the priority operator "θ" [6]. For a given irreflexive partial order $\gg$ 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 $b \gg a$ is possible in that state. If $B_a = \{b \mid b \gg a\}$, then θ is defined in a natural fashion by the following GSOS rules, one for each action a, where expressions of the form $X \stackrel{b}{\not\rightarrow}$ in the premises are called *negative premises*:

$$\frac{X \stackrel{a}{\rightarrow} X' \quad \{X \stackrel{b}{\not\rightarrow}\}_{b \in B_a}}{\theta(X) \stackrel{a}{\rightarrow} \theta(X')}$$

The second procedure in [2], also described in [1], produces the following axioms for θ where the basic operators of CCS, namely "+", prefixing and "**0**", are used. Since a typical rule for θ may have several copies of the argument X in the premises

an auxiliary binary operator "$\triangle$", defined below, is used [2].

$$\frac{X \xrightarrow{a} X' \quad \{Y \overset{b}{\not\to}\}_{b \in B_a}}{X \triangle Y \xrightarrow{a} \theta(X')}$$

The following axioms for θ consist of the axiom that makes copies of X and uses the auxiliary operator $\triangle$, and the axioms for $\triangle$ consisting of the distributivity axiom, peeling axioms and inaction axioms:

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

The priority operator can be defined equivalently, and perhaps more intuitively, by *positive* GSOS rules equipped with an *ordering* to represent the priority order on actions: the ordering has the corresponding effect to negative premises in rules. This is the idea behind the *Ordered* SOS format [40]. The rules for the OSOS version of θ are, one for each a,

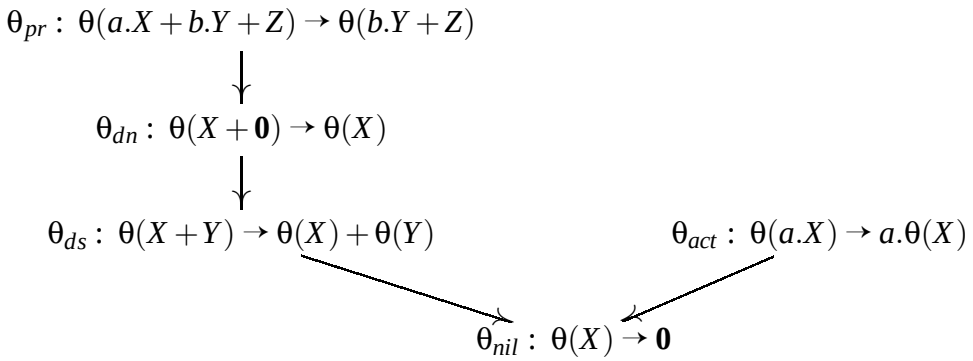
$$\frac{X \xrightarrow{a} X'}{\theta(X) \xrightarrow{a} \theta(X')} \quad r_a$$

and the ordering $>$ is such that $r_b > r_a$ whenever $b \gg a$. The ordering prescribes that rule r_a can be applied to derive transitions of $\theta(p)$ if no higher priority rule, e.g. r_b, can be applied to $\theta(p)$. This suggests an axiomatisation procedure: derive the axioms from the SOS rules similarly to [2,1], and then "order" them appropriately according to the ordering on the SOS rules. More precisely, we orientate the axioms from left to right to obtain the rewrite rules, then define a *priority* ordering which is an irreflexive partial order (irreflexive and transitive) on the rewrite rules, and then introduce a new type of rewrite rule to deal with the priority ordering. What we obtain is an example of a *Priority Rewrite System*, or PRS for short, originated by Baeten, Bergstra, Klop and Weijland [7]. Our procedure generates the following PRS for the operator θ. We have one rewrite rule θ^b_{pr} for each pair of a and b such

that $b \gg a$, and one θ^a_{act} rule for each action a:

$$\begin{aligned}
\theta^b_{pr} &: \ \theta(a.X + b.Y + Z) \to \theta(b.Y + Z) \\
\theta_{dn} &: \ \theta(X + \mathbf{0}) \to \theta(X) \\
\theta_{ds} &: \ \theta(X + Y) \to \theta(X) + \theta(Y) \\
\theta^a_{act} &: \ \theta(a.X) \to a.\theta(X) \\
\theta_{nil} &: \ \theta(X) \to \mathbf{0}
\end{aligned}$$

The priority ordering on the rewrite rules is defined as follows: $\theta^b_{pr} \succ \theta_{dn}$ for all rewrite rules θ^b_{pr}, $\theta_{dn} \succ \theta_{ds}$ and $\{\theta_{ds},\} \cup \{\theta^a_{act} \mid \text{ all } a\} \succ \theta_{nil}$. We can represent this ordering more pictorially. Below, $r \succ r'$ if and only if there is an arrow from r to r':



Note, that we have fewer rewrite rules (schemas) than the axioms (schemas) above, and no need for the auxiliary $\triangle$.

Our PRSs are sound for bisimulation, meaning that closed terms can only be rewritten to bisimilar closed terms, and they are head normalising. The main technical result here is Lemma 5.4 which describes how to construct the auxiliary term and the auxiliary rewrite rule. For OSOS PLs generating finite behaviours, in our case linear and syntactically well-founded OSOS PLs, the generated PRSs are also strongly normalising (terminating) and confluent. The proof of termination (Theorem 6.4) uses novel dependency pairs and dependency graphs techniques, and generalises the proof of termination by Bosscher in [14]. Finally, for the mentioned subclass of OSOS PLs, the generated PRSs are complete for bisimulation: if two closed terms are bisimilar, then they are reducible to the unique, modulo the associativity and commutativity of $+$, normal form.

The paper is organised as follows. In Section 2 we recall the definitions of OSOS PLs and bisimulation, and Section 3 presents the basics of term rewriting, rewriting modulo associativity and commutativity of $+$, and rewriting with priority order. In Section 4 we introduce our basic PL and construct a PRS for it. The PRS is strongly normalising, confluent, and sound and complete for bisimulation. Section

5 presents a procedure for generating PRSs for arbitrary OSOS process languages. Termination of PRSs is discussed and a termination result for syntactically well-founded and linear OSOS PLs is given in Section 6. Section 7 contains confluence and completeness results for bisimulation. The last section contains conclusions and ideas for possible extensions.

2 Preliminaries

This section recalls some results concerning processes, labelled transition systems, bisimulation, and the GSOS and OSOS formats. We assume a knowledge of basic definitions and results for PLs as in [15,27,10] and for SOSs as in [13,18].

2.1 Transition System and Bisimulation

Definition 2.1 A labelled transition system, 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 $p, q, r, s, t, \ldots$. The set Act is a finite set of actions and it is ranged over by a, b, c and their subscripted versions. The action τ is the silent action but we do not treat it any differently from other actions. We permit Act to have a structure: for example Act may consist of action labels and co-labels as in CCS [27]. We will use the following abbreviations. We write $p \xrightarrow{a} q$ 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 $p \xrightarrow{a} q$ will be called *transitions*. We write $p \xrightarrow{a}$ when there is some q such that $p \xrightarrow{a} q$, and $p \xnrightarrow{a}$ otherwise.

We recall the definition of bisimulation [30,27]:

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 such that pRq and all $a \in \mathsf{Act}$, the following properties hold.

$$p \xrightarrow{a} p' \text{ implies } \exists q'.(q \xrightarrow{a} q' \text{ and } p'Rq')$$

$$q \xrightarrow{a} q' \text{ implies } \exists p'.(p \xrightarrow{a} p' \text{ and } p'Rq')$$

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

The OSOS format [40] is an alternative to the GSOS format [13]. The reader can find the motivation for the OSOS format and many examples of its application in [40]. It is important to state that the OSOS format is as expressive as the GSOS format [40,41]. Before we recall the definitions of the formats we introduce several notions and notations.

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 union 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 operator. The set of *open terms* over Σ with variables in $V \subseteq \mathsf{Var}$, denoted by $\mathbb{T}(\Sigma, V)$, is ranged over by $t, t', \ldots$. $Var(t) \subseteq \mathsf{Var}$ is the set of variables in a term t. 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 $C[X_1, \ldots, X_n]$ is a member of $\mathbb{T}(\Sigma, \{X_1, \ldots, X_n\})$, i.e. an open term that contain at most variables X_i for $1 \leq i \leq n$. 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. 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. Given any binary relation R on closed terms and $\boldsymbol{p}$ and $\boldsymbol{q}$ of length n, we will write $\boldsymbol{p}R\boldsymbol{q}$ to mean $p_i R q_i$ for all $1 \leq i \leq n$. Moreover, instead of $f(X_1, \ldots, X_n)$ we will often write $f(\boldsymbol{X})$ when the arity of f is understood. An equivalence relation $\approx$ over a PL over Σ is a *congruence* if $\boldsymbol{p} \approx \boldsymbol{q}$ implies $C[\boldsymbol{p}] \approx C[\boldsymbol{q}]$ for all $\boldsymbol{p}$ and $\boldsymbol{q}$ of length n and all Σ contexts $C[\boldsymbol{X}]$ with n holes.

A *closed substitution* is a mapping $\mathsf{Var} \to \mathrm{T}(\Sigma)$. Closed substitutions are ranged over by ρ, ρ' and σ; they extend to $\mathbb{T}(\Sigma, \mathsf{Var}) \to \mathrm{T}(\Sigma)$ mappings in a standard way. For t with $Var(t) \subseteq \{X_1, \ldots, X_n\}$ we write $t[p_1/X_1, \ldots, p_n/X_n]$ or $t[\boldsymbol{p}/\boldsymbol{X}]$ to mean t with each X_i replaced by p_i, where $1 \leq i \leq n$.

Definition 2.3 [13] A GSOS rule is an expression of the form

$$\frac{\{\, X_i \xrightarrow{a_{ij}} Y_{ij} \,\}_{i \in I, j \in J_i} \quad \{\, X_k \overset{b_{kl}}{\not\to} \,\}_{k \in K, l \in L_k}}{f(\boldsymbol{X}) \xrightarrow{a} C[\boldsymbol{X}, \boldsymbol{Y}],} \tag{1}$$

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 rule of the form (1). Operator f is the *operator* of r and $rules(f)$ is the set of all rules with the operator f. Expressions $t \xrightarrow{a} t'$ and $t \overset{a}{\not\to}$, where $t, t' \in \mathbb{T}(\Sigma, V)$,

are called *transitions* and *negative transitions* respectively. Transitions are ranged over by T and T'. If transition T is $X \xrightarrow{a} X'$, we will sometime use the notation $\neg T$ to stand for $X \xrightarrow{a} \not\rightarrow$. A (negative) transition which involves only closed terms is called a *closed* (negative) transition. The set of transitions and negative transitions above the horizontal bar in r is called the *premises* of r, and is written as $pre(r)$. The transition below the bar in r is the *conclusion*, written as $con(r)$. Action a in the conclusion of r is the *action* of r, written as $act(r)$, and $f(\boldsymbol{X})$ and $C[\boldsymbol{X},\boldsymbol{Y}]$ are the *source* and *target* of r, respectively. The i-th argument X_i is *active* in r if $X_i \xrightarrow{a_{ij}} Y_{ij}$ or $X_i \xrightarrow{b_{il}} \not\rightarrow$ is a premise of r. The set of all i such that X_i is active in r is denoted by $active(r)$. Moreover, the i-th argument of f is *active* if $i \in active(r')$ for some rule r' for f.

Definition 2.4 A *positive* GSOS rule (transition rule, or OSOS rule, or simply a rule) is a GSOS rule with $K = \emptyset$. With the notation as in Definition 2.3, it has the following form:

$$\frac{\{\ X_i \xrightarrow{a_{ij}} Y_{ij}\ \}_{i\in I, j\in J_i}}{f(\boldsymbol{X}) \xrightarrow{a} C[\boldsymbol{X},\boldsymbol{Y}].} \tag{2}$$

Next, we recall the notion of *ordering* on rules [40]. It is a new feature which allows the user to control the order of application of OSOS rules (positive GSOS rules) when deriving transitions of process terms.

An ordering on OSOS rules for operator f, $>_f$, is a binary relation over the rules for f. For the purpose of this paper we assume without loss of generality that orderings are *irreflexive* (i.e. $r > r$ never holds) and *transitive*. In general there are situations, which are described and motivated in [40], where non-transitive or not irreflexive relations are useful orderings on rules. Expression $r >_f r'$ is interpreted as r having higher priority than r' when deriving transitions of terms with f as the outermost operator. Given Σ, the relation $>_\Sigma$, or simply $>$ if Σ is known from the context, is defined as $\bigcup_{f\in\Sigma} >_f$. We will denote $\{r' \mid r' > r\}$ as $higher(r)$, and generalise it to $higher(R)$ for sets of OSOS rules R.

Definition 2.5 A GSOS PL is a tuple (Σ, A, R), where Σ is a finite set of operators, $A \subseteq \mathsf{Act}$, R is a finite set of GSOS rules for operators in Σ such that all actions mentioned in the rules belong to A. An operator of a GSOS PL is called a GSOS operator.

An *Ordered SOS* (or OSOS, for short) PL is a tuple $(\Sigma, A, R, >)$, where Σ is a finite set of operators, $A \subseteq \mathsf{Act}$, R is a finite set of OSOS rules for operators in Σ, written as $rules(\Sigma)$, such that all actions mentioned in the rules belong to A, and $>$ is an ordering on $rules(\Sigma)$. An operator of an OSOS PL is called an OSOS operator.

Given an OSOS process language $G = (\Sigma, A, R, >)$, we associate a unique transition

relation $\rightarrow$ with G. The details are given in [40]. Having the transition relation for G we easily construct $(\mathrm{T}(\Sigma), A, \rightarrow)$, the LTS for G. Bisimulation is defined over this LTS as in Definition 2.2. Since GSOS and OSOS are equally expressive, namely every GSOS process language can be equivalently given as an OSOS process language and vice versa [40], bisimulation is a congruence for all OSOS PLs.

An OSOS PL H is a *disjoint extension* of an OSOS PL G, written as $G \leq H$, if the signature, the rules and the orderings of H include those of G, and H introduces no new rules and orderings for the operators in G.

Finally, we give two examples of process operators that have natural and intuitive definitions in terms of OSOS rules.

Definition 2.6 Let r be a rule for an OSOS operator f such that $pre(r) = \{X_i \stackrel{a_{ij}}{\rightarrow} Y_{ij} \mid i \in I, j \in J_i\}$. We say that rule r *applies* to $f(\boldsymbol{u})$ if and only if the premises of r are valid for $\boldsymbol{u}$, namely $u_i \stackrel{a_{ij}}{\rightarrow}$ for all relevant i and j. Rule r is *enabled* at $f(\boldsymbol{u})$ if and only if r applies to $f(\boldsymbol{u})$ and no rules in $higher(r)$ apply to $f(\boldsymbol{u})$.

Example 2.7 Consider the OSOS and GSOS definitions of the sequential composition operator ";":

$$\frac{X \stackrel{a}{\rightarrow} X'}{X;Y \stackrel{a}{\rightarrow} X';Y}\ r_{a*} \quad > \quad \frac{Y \stackrel{b}{\rightarrow} Y'}{X;Y \stackrel{b}{\rightarrow} Y'}\ r_{*b} \qquad\qquad \frac{\{X \stackrel{a}{\not\rightarrow}\}_{a \in \mathsf{Act}}\ Y \stackrel{b}{\rightarrow} Y'}{X;Y \stackrel{b}{\rightarrow} Y'}\ r_{nb}$$

Rules r_{a*} and r_{*b}, for all actions a and b, together with $>$ defined by $r_{a*} > r_{*b}$, for all a and b, comprise the OSOS formulation. Rules r_{a*} and r_{nb}, for all a and b, form the GSOS definition.

Consider processes p and q with $q \stackrel{b}{\rightarrow}$. Using the OSOS definition, process $p;q$ can perform an initial action b of q, inferred by r_{*b}, if all rules r_{a*} are not applicable. This occurs when the premises of these rules are not valid: i.e. $p \stackrel{a}{\not\rightarrow}$ for all $a \in \mathsf{Act}$. So, the ordering on the OSOS rules for ; has the same effect as GSOS rules r_{nb} with the negative premises $\{X \stackrel{a}{\not\rightarrow}\}_{a \in \mathsf{Act}}$.

Example 2.8 Consider Hennessy and Regan's *Temporal Process Language* (TPL) [19]. It has a delay operator "$\lfloor\ \rfloor(\ )$" defined by the following GSOS rules, where a is any action except τ and the action σ denotes the passage of one time unit. So, the first rule below is really a rule schema for all $a \neq \tau$.

$$\frac{X \stackrel{a}{\rightarrow} X'}{\lfloor X \rfloor(Y) \stackrel{a}{\rightarrow} X'} \qquad \frac{X \stackrel{\tau}{\rightarrow} X'}{\lfloor X \rfloor(Y) \stackrel{\tau}{\rightarrow} X'} \qquad \frac{X \stackrel{\tau}{\not\rightarrow}}{\lfloor X \rfloor(Y) \stackrel{\sigma}{\rightarrow} Y}$$

The OSOS formulation of $\lfloor\ \rfloor(\ )$ is straightforward. The OSOS rules are

$$\frac{X \xrightarrow{a} X'}{\lfloor X \rfloor(Y) \xrightarrow{a} X'} \qquad \frac{X \xrightarrow{\tau} X'}{\lfloor X \rfloor(Y) \xrightarrow{\tau} X'}\ \tau^1 \qquad \lfloor X \rfloor(Y) \xrightarrow{\sigma} Y\ \ \sigma_\emptyset$$

and the ordering is $\tau^1 > \sigma_\emptyset$. The parallel composition operator '$\|$' of TPL a timed extension of the CCS parallel with the following non-GSOS rule:

$$\frac{X \xrightarrow{\sigma} X' \quad Y \xrightarrow{\sigma} Y' \quad X \parallel Y \overset{\tau}{\not\to}}{X \parallel Y \xrightarrow{\sigma} X' \parallel Y'}$$

The rule requires that $p \parallel q$ can pass time if both p and q can pass time and are stable and cannot communicate. The operator has the following OSOS formulation. Its rules are precisely the CCS rules (we only display communication rule schema $r_{a\bar{a}}$) together with the following timed rule r_σ,

$$\frac{X \xrightarrow{a} X' \quad Y \xrightarrow{\bar{a}} Y'}{X \parallel Y \xrightarrow{\tau} X' \parallel Y'}\ r_{a\bar{a}} \qquad \frac{X \xrightarrow{\sigma} X' \quad Y \xrightarrow{\sigma} Y'}{X \parallel Y \xrightarrow{\sigma} X' \parallel Y'}\ r_\sigma$$

which is placed below all the rules for $\parallel$ with the action τ, namely the two τ-rules and all the communications rules $r_{a\bar{a}}$. The GSOS formulation of the operator is less natural: see [41].

Most of the process operators that are definable by GSOS rules with negative premises have OSOS formulations which are as natural and efficient as those of the sequential composition and the priority operators discussed above. The examples are priority choice from Section 4, action refinement operator [40], the hiding operator of ET-LOTOS [24], several delay operators [29,19,9], and several timed extensions of traditional operators: for example parallel composition of *TPL*, and hiding and sequential composition of CSP [33,35].

2.3 Classes of GSOS and OSOS Operators

The axiomatisation algorithms in [2] produce several types of laws (axioms) for GSOS operators depending on the form of their SOS definitions. Three types of SOS definitions, and hence three classes of operators, are defined: *smooth*, *distinctive* and *discarding*. Our PRS algorithm relies also on partitioning OSOS operators into similar classes. We identify two classes: *free of implicit copies* operators and *simply distinctive* operators. In order to compare the algorithms for the GSOS PLs and the presented algorithm for the OSOS PLs we state and compare the definitions of the mentioned classes of operators.

A GSOS rule is smooth [2] if it has the form

$$\frac{\{\ X_i \xrightarrow{a_i} Y_i\ \}_{i\in I} \quad \{\ X_k \overset{b_{kl}}{\not\rightarrow}\ \}_{k\in K, l\in L_k}}{f(X_1,\ldots,X_n) \xrightarrow{a} C[\boldsymbol{X},\boldsymbol{Y}],}$$

where I and K are distinct sets and $I\cup K=\{1,\ldots,n\}$, and no X_i appears in $C[\boldsymbol{X},\boldsymbol{Y}]$ when $i\in I$. A GSOS operator is smooth if all its rules are smooth.

Multiple occurrences of process variables in the (positive) premises and in the target of SOS rules are called *copies*. They are either *explicit* or *implicit* copies [37,40]. Given a rule r as in Definition 2.4, explicit copies are the multiple occurrences of variables Y_{ij} in the target $C[\boldsymbol{X},\boldsymbol{Y}]$ and the multiple occurrences of 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 of r and the occurrences, not necessarily multiple, of variables X_i in $C[\boldsymbol{X},\boldsymbol{Y}]$ for $i\in I$. Consider the following rule r_h:

$$\frac{X_1 \xrightarrow{a_{11}} Y_{11} \quad X_1 \xrightarrow{a_{12}} Y_{12} \quad X_2 \xrightarrow{a_{21}} Y_{21}}{h(X_1,X_2,X_3,X_4) \xrightarrow{a} g(X_2,X_3,X_3,X_4,Y_{11},Y_{11})}$$

The multiple occurrences of X_1 in the premises of r_h are implicit copies, and the occurrence of X_2 in the target is also an implicit copy (of X_2). The occurrences of X_3 and Y_{11} in the target are explicit copies. There are no implicit and no explicit copies of X_4 in r_h since X_4 does not appear in the premises.

Definition 2.9 A rule with no implicit copies is *free of implicit copies*. An OSOS operator is *free of implicit copies* if its rules are free of implicit copies.

We notice that smooth GSOS rules can be defined using the notion of implicit copies: A GSOS rule of the form (1) is smooth if it has no implicit copies, I and K are distinct sets and $I\cup K=\{1,\ldots,n\}$. Consequently, the following results hold.

- If a GSOS operator is smooth, then there is an OSOS formulation of the operator which is free of implicit copies [40].
- The converse is not valid: There are non-smooth GSOS operators whose OSOS formulations are free of implicit copies.

The second result holds for non-smooth GSOS operators which have rules with arguments that appear in both positive and negative premises. The priority operator θ and the timed version of the parallel operator of TPL (Example 2.8) are examples of GSOS operators which are not smooth and which have OSOS formulations that are free of implicit copies. Further examples are the hiding operators of the discrete time versions of CSP [35] and ET-LOTOS [24] given in [41] and recalled in Example 5.11.

The next class of GSOS operators used by the axiomatisation procedures in [2] are the distinctive operators: a smooth GSOS operator f is distinctive if, for each argument i, the argument either appears in positive premises of 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 positive premise but with a different action. The prefixing, renaming and restriction operators of CCS are distinctive operators, whereas the choice operator and the parallel operator of CCS, and sequential composition operators are not distinctive. We shall use a similar notion:

Definition 2.10 An OSOS operator f which is free of implicit copies is *simply distinctive* if the ordering on its rules is empty and, for each argument i, the argument either appears in premises 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.

3 Term Rewriting Systems

We recall the basic notions of term rewriting [22,5]. A Term Rewriting System (TRS) $\mathcal{R}$ 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 $V \subseteq \mathsf{Var}$ 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 $C[X]$ 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 reflexive and transitive 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* (*terminating*) 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. Recall, that s and t are

joinable, written as $t \downarrow s$, if they have a common reduct u, namely $s \twoheadrightarrow u$ and $t \twoheadrightarrow u$. 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 $D[X]$ 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 [22] that states that if a TRS is strongly normalising, then it is confluent if and only if all its critical pairs are convergent.

3.1 Rewriting modulo AC

We assume a knowledge of basic notions of term rewriting as, for example, in [22]. 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:

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

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, t' and s such that $t \in [t']_{AC}$ if $t' \to s$, then we shall write $t \to_{AC} s$ and $[t]_{AC} \to_{AC} [s]_{AC}$. We define $t \twoheadrightarrow_{AC} s$ and $[t]_{AC} \twoheadrightarrow_{AC} [s]_{AC}$ as the appropriate transitive reflexive closures of $\to_{AC}$. The internal reductions of $t \twoheadrightarrow_{AC} s$ and $[t]_{AC} \twoheadrightarrow_{AC} [s]_{AC}$ are defined in the corresponding way to the internal reductions of $\twoheadrightarrow$. Henceforth, we drop all subscripts AC.

Example 3.1 Consider a fragment of CCS with the signature $\Sigma = \{(\mathbf{0}, 0), (+, 2)\} \cup \{(a., 1) \mid a \in \mathsf{Act}\}$, where $\mathbf{0}$ is the deadlocked process operator, $a.$ are the prefixing with actions a operators, for all $a \in \mathsf{Act}$, and $+$ the CCS choice operator. The closed terms over Σ represent finite trees. Let (Σ, R) be a TRS with the following set R of

reduction rules:

$$r_1 : X + \mathbf{0} \rightarrow X$$
$$r_2 : X + X \rightarrow X$$

Term $a.X + (a.X + \mathbf{0})$ reduces to $a.X$ as follows: $a.X + (a.X + \mathbf{0}) \rightarrow_{r_1} a.X + a.X \rightarrow_{r_2} a.X$. There is another reduction modulo *AC* to $a.X$: $a.X + (a.X + \mathbf{0}) = (a.X + a.X) + \mathbf{0} \rightarrow_{r_2} a.X + \mathbf{0} \rightarrow_{r_1} a.X$. Hence, $[a.X + (a.X + \mathbf{0})] \twoheadrightarrow [a.X]$.

(Σ, 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 r_1 and r_2 have a simple overlap which replaces X with $\mathbf{0}$. Now, we have $\mathbf{0} + \mathbf{0} \rightarrow_{r_1} \mathbf{0}$ and $\mathbf{0} + \mathbf{0} \rightarrow_{r_2} \mathbf{0}$. Hence, there is only one critical pair $([\mathbf{0}], [\mathbf{0}])$, and it is joinable.

3.2 Priority Rewriting

As transition rules for process operators can be equipped with orderings that indicate which transition rules to apply first, reduction rules can also have an ordering associated with them. This ordering, called *priority* order, specifies the order in which rewrite rules are to be used to rewrite a term. This is illustrated by the following simple example.

Example 3.2 The TRS from Example 3.1 is now equipped with a priority order $\succ$ defined by $r_1 \succ r_2$. As before, $a.X + (a.X + \mathbf{0}) \twoheadrightarrow [a.X]$ because $a.X + (a.X + \mathbf{0}) \rightarrow_{r_1} a.X + a.X$, and since $a.X$ cannot be reduced to $\mathbf{0}$, $a.X + a.X$ then reduces to $a.X$ by rule r_2. However, the second reduction from Example 3.1 is not correct (intended) in this new setting. After $a.X + (a.X + \mathbf{0}) = (a.X + a.X) + \mathbf{0}$ we see that both r_1 and r_2 can be applied; but since r_1 has priority over r_2 we must apply r_1: $(a.X + a.X) + \mathbf{0} \rightarrow_{r_1} a.X + a.X$. Now, only r_2 can be applied.

Next, consider term $t \equiv (a.X + \mathbf{0}) + (a.X + \mathbf{0})$. The term is an r_2-redex, it is not an r_1-redex although it contains r_1-redexes. We may wish to reduce the term with r_2 ahead of r_1. This is not intended in the new setting: we must either use higher priority r_1 to reduce subterms $a.X + \mathbf{0}$ to $a.X$ first, or use *AC* to convert t to r_1-redex $((a.X + \mathbf{0}) + a.X) + \mathbf{0}$ that can be reduced as follows:

$$[((a.X + \mathbf{0}) + a.X) + \mathbf{0}] \rightarrow_{r_1} [(a.X + a.X) + \mathbf{0}] \rightarrow_{r_1} [a.X + a.X] \rightarrow_{r_2} [a.X].$$

In general, a rewrite rule r_2 with a lower priority than r_1 can be applied to term t in favour of r_1, if no **internal** reduction (reduction sequence leaving head operator

unaffected) modulo *AC* of t can produce a contractum that is an r_1-redex. We recall the basic notions of term rewriting with priority [7,34,32].

Definition 3.3 A Priority Rewrite System, or PRS for short, is a tuple $(\Sigma, \mathrm{T}, \succ)$, where (Σ, T) is a TRS and $\succ$ is a partial order on T called *priority order*. Let $\mathcal{P} = (\Sigma, \mathrm{T}, \succ)$ be a PRS, and let R be a set of rewrites for $\mathcal{P}$, namely closed substitutions of reduction rules of $\mathcal{P}$. The rewrite $r : t \to s$ is *correct* with respect to R (modulo *AC*) if there is no internal reduction $[t] \twoheadrightarrow_R [t']$ and no rule $r' : t' \to s' \in R$ such that $r' \succ r$. R is *sound* if all its rewrites are correct w.r.t. R. R is *complete* if it contains all rewrites of $\mathcal{P}$ which are correct w.r.t. R. $\mathcal{P}$ is *well-defined* if it has a unique sound and complete rewrite set; this set is called the *semantics* of $\mathcal{P}$.

A PRS is well-defined if the underlying TRS is strongly normalising [7]. Hence, the PRS from Example 3.2 is well-defined. It is also strongly normalising by the result below which follows by a simple proof by contradiction.

Proposition 3.4 *If the underlying TRS of a PRS is strongly normalising modulo AC, then the PRS is well-defined and strongly normalising modulo AC.*

The PRS in Example 3.2 is confluent because, although r_1 and r_2 overlap, the priority order disables r_2, thus $\mathbf{0} + \mathbf{0} \to_{r_1} \mathbf{0}$ is the only reduction from $\mathbf{0} + \mathbf{0}$. Hence, there are no critical pairs.

4 Basic Process Language

In this section we define a simple process language which is an extension of the process language for finite trees from Example 3.1. It contains a new operator, called *priority choice*, which is denoted by "$\rhd$". We introduce a PRS for this language and show that it is sound and complete for bisimulation. This language and its PRS are the foundations on which we shall build PRSs for arbitrary OSOS PLs; the language plays the rôle corresponding to that of FINTREE in [2].

Definition 4.1 *Basic Process Language* B is an OSOS PL $(\Sigma_{\mathsf{B}}, A, R, >)$, where $\Sigma_{\mathsf{B}} = \Sigma_0 \cup \Sigma_1 \cup \Sigma_2$ with $\Sigma_0 = \{(\mathbf{0}, 0)\}$, $\Sigma_1 = \{(a., 1) \mid a \in A\}$ and $\Sigma_2 = \{(+, 2), (\rhd, 2)\}$, $A \subset_{fin} \mathsf{Act}$, and R and $>$ are the set of transition rules and the ordering on transition rules, respectively. The rule schemas for the prefixing operators and the two choice operators are

$$a.X \xrightarrow{a} X$$

$$\frac{X \xrightarrow{a} X'}{X + Y \xrightarrow{a} X'} \qquad \frac{Y \xrightarrow{a} Y'}{X + Y \xrightarrow{a} Y'} \qquad \frac{X \xrightarrow{a} X'}{X \rhd Y \xrightarrow{a} X'}\, r_{a*} \qquad \frac{Y \xrightarrow{c} Y'}{X \rhd Y \xrightarrow{c} Y'}\, r_{*c}$$

$$
\begin{array}{ll}
+_{dn} : X + \mathbf{0} & \to X \\
+_{ice} : X + X & \to X \\
\rhd_{dn1} : (X + \mathbf{0}) \rhd Z & \to X \rhd Z \\
\rhd_{ds1} : (X + Y) \rhd Z & \to X \rhd Z + Y \rhd Z \\
\rhd_{act} : a.X \rhd Y & \to a.X \\
\rhd_{nil} : X \rhd Y & \to Y
\end{array}
$$

$$
+_{dn} \succ +_{ice} \text{ and } \rhd_{dn1} \succ \rhd_{ds1} \text{ and } \{\rhd_{ds1}, \rhd_{act}\} \succ \rhd_{nil}
$$

Fig. 1. Rewrite rules and the priority order for B.

and the ordering is $r_{a*} > r_{*c}$ for all actions a, c. The prefixing operators bind stronger than $\rhd$, which in turn binds stronger than $+$.

B generates the LTS $B = (\mathrm{T}(\Sigma_{\mathsf{B}}), A, \to)$. Bisimulation over B is defined accordingly. Let $\mathcal{B}$ be the PRS for B defined in Figure 1. Notice that reduction rules $+_{dn}, +_{ice}$ (*i*dempoten*ce*) and $\rhd_{act}$ are sound for bisimulation on their own but $\rhd_{ds1}$ (distributivity over 1st argument) is not sound on its own. For let $\sigma X = \mathbf{0}$, $\sigma Y = a.\mathbf{0}$ and $\sigma Z = b.\mathbf{0}$. Then $\sigma((X+Y) \rhd Z) \sim a.\mathbf{0}$, $\sigma(X \rhd Z + Y \rhd Z) \sim a.\mathbf{0} + b.\mathbf{0}$ and, clearly, $a.\mathbf{0} \not\sim a.\mathbf{0} + b.\mathbf{0}$. However, putting $\rhd_{ds1}$ below $\rhd_{dn1}$ solves this problem as $\rhd_{ds1}$ can only be applied when neither σX nor σY reduces to $\mathbf{0}$.

Definition 4.2 Let $G = (\Sigma, A, S, >)$ be an OSOS PL. Let $\mathcal{P} = (\Sigma, \mathrm{T}, \succ)$ be a well-defined PRS with its unique sound and complete rewrite set R. A rewrite $t \to s$ of R, where t and s are closed Σ terms, is *sound for bisimulation* if $t \sim s$. A rewrite rule $\mathrm{T} \ni r_0 : l \to r$ is sound for bisimulation if every r_0-rewrite, which is correct with respect to the semantics of $\mathcal{P}$, is sound for bisimulation. $\mathcal{P}$ is sound for bisimulation if all its rewrite rules are. The set R is *complete for bisimulation* if whenever $t \sim s$, then $t \downarrow s$. $\mathcal{P}$ is complete for strong bisimulation if its rewrite set R is.

Theorem 4.3 *$\mathcal{B}$ is strongly normalising and confluent modulo AC.*

PROOF. To show strong normalisation of $\mathcal{B}$ it is enough to prove that the underlying TRS of $\mathcal{B}$ is strongly normalising modulo *AC*. We select polynomial interpretations as follows: interpret $\mathbf{0}$, $\alpha.X$, $X+Y$ and $X \rhd Y$ as 2, $2X$, $X+Y+1$ and XY. It can be checked that for each rewrite rule in Figure 1 this polynomial interpretation makes the right-hand side strictly smaller than the left-hand side for natural numbers greater than one. Since equations *AC* are also satisfied by this polynomial interpretation the considered TRS is strongly normalising modulo *AC*.

Since the PRS $\mathcal{B}$ is strongly normalising it is sufficient to show that all critical pairs are joinable in order to obtain confluence. There are only three critical pairs: $\{(\mathbf{0} \rhd Z, \mathbf{0} \rhd Z),\ (X \rhd Z, X \rhd Z),\ (X \rhd Z, X \rhd Z + X \rhd Z)\}$. We easily see that they are joinable. There are other overlaps between the rules of $\mathcal{B}$, for example the over-

lap between $+_{dn}$ and $\rhd_{ds1}$. This overlap would seem to lead to the critical pair $(X \rhd Z,\ X \rhd Z + \mathbf{0} \rhd Z)$, which clearly is not convergent and not sound for bisimulation. However, since $\rhd_{dn1} \succ \rhd_{ds1}$ the term $(X + \mathbf{0}) \rhd Z$ can only be rewritten with $\rhd_{dn1}$ to $X \rhd Z$, and not with $\rhd_{ds1}$. Such overlaps do not produce critical pairs: they show how priority order decreases the nondeterminism that is inherent in term rewriting. □

Normal forms and *head normal forms* (abbreviated to nf and hnf, respectively, to distinguish them from the normal forms in term rewriting) over a PL that extends B are defined as follows: $\mathbf{0}$ is in nf; if t is in nf, then $a.t$ is in nf for all relevant a; and if t and s are in nf, then $t + s$ is in nf unless t and s are syntactically equal or either s or t is $\mathbf{0}$. For head normal forms: $\mathbf{0}$ and $a.t$ are in hnf for any term t, and if t and s are in hnf, then $t + s$ is in hnf unless t and s are syntactically equal or either s or t is $\mathbf{0}$.

Theorem 4.4 *$\mathcal{B}$ is sound and complete for bisimulation.*

PROOF. Since $\mathcal{B}$ is strongly normalising it is well-defined [7]. Let B be the rewrite set of $\mathcal{B}$. The soundness for bisimulation of the rewrite rules in Figure 1 is clear except possibly for $\rhd_{ds1}$ and $\rhd_{nil}$. Without the priority order these rewrite rules are clearly unsound. In general, the ordering $\rhd_{dn1} \succ \rhd_{ds1}$ ensures that $\rhd_{ds1}$ can be applied only when neither the term substituted for X nor the term substituted for Y can be reduced to $\mathbf{0}$. It clear that for such substitutions $\rhd_{ds1}$ is sound for bisimulation. Finally, $\rhd_{nil}$ can be applied to reduce a term $p \rhd q$ if p is not reducible to either a sum of subterms or an action prefixed term. Hence, p must be $\mathbf{0}$.

For completeness assume $p \sim q$ for closed terms p and q over $\mathcal{B}$. By Theorem 4.3 we know that, for every closed term t over B, there exists a unique normal form s such that $t \twoheadrightarrow_B s$. Also, we easily show that a closed term over B is a normal form w.r.t. rewriting if and only if it is in nf. Hence, there are terms p' and q' such that p B-reduces to p', q B-reduces to q', and p' and q' are in nf. Since rewriting is sound for bisimulation we get $p' \sim q'$. Now, we can show that p' and q', which are in nf, are equal modulo AC. Hence, $p' \downarrow q'$ and thus $p \downarrow q$ as required. □

In the next section we show how to generate PRSs for arbitrary well-founded OSOS PLs that extend disjointly our language B. The proof of completeness of such PRSs uses the above completeness result for B.

5 Rewrite Rules for OSOS Operators

Operators of an arbitrary OSOS PL can be partitioned, according to their OSOS definitions, into three disjoint sets: (1) operators that are not free of implicit copies, (2) operators that are free of implicit copies and not simply distinctive, and (3) simply distinctive operators. We describe the type of rewrite rules and priority orderings for each of these types of operators (and auxiliary operators) in the following three subsections. Finally, we introduce our algorithm for generating PRS for arbitrary PL in the OSOS format.

5.1 Operators with implicit copies

If an OSOS operator (f,n) is not free of implicit copies, then we can construct a free of implicit copies OSOS operator (f^c,m), with $m > n$, that does the job of f.

Lemma 5.1 *Let G be an OSOS PL with signature Σ. Let $\mathcal{P} = (\Sigma, \mathrm{R}, \succ)$ be a well-defined PRS for G that is sound for bisimulation. Suppose $(f,n) \in \Sigma$ is an operator not free of implicit copies. Then, there is*

- *a disjoint extension of G' of G with a free of implicit copies operator (f^c,m) such that $m > n$,*
- *a PRS $\mathcal{P}' = (\Sigma \cup \{f^c\}, \mathrm{R} \cup \{f_{copy}\}, \succ)$ with the new* **copying** *rewrite rule below, where $\boldsymbol{X}$ is some vector of n distinct variables and $\boldsymbol{Y}$ is a vector of m variables from $\boldsymbol{X}$,*

$$f_{copy}:\ f(\boldsymbol{X}) \rightarrow f^c(\boldsymbol{Y}),$$

and the PRS $\mathcal{P}'$ is sound for bisimulation.

PROOF. Correspondingly as for GSOS operators which are not smooth due to having implicit copies: see proofs of Lemmas 5.1 and 5.2 in [2]. □

As an example consider operator $(h,4)$ from Section 2.4. The operator has implicit copies of its first two arguments and the operator h^c, the free of implicit copies version of h produced by Lemma 5.1, uses extra two arguments as follows:

$$\frac{X_1^1 \xrightarrow{a_{11}} Y_{11} \quad X_1^2 \xrightarrow{a_{12}} Y_{12} \quad X_2^1 \xrightarrow{a_{21}} Y_{21}}{h^c(X_1^1,X_1^2,X_2^1,X_2^2,X_3,X_4) \xrightarrow{a} g(X_2^2,X_3,X_3,X_4,Y_{11},Y_{11})}$$

The copying rewrite rule for h is $h(X_1,X_2,X_3,X_4) \rightarrow h^c(X_1,X_1,X_2,X_2,X_3,X_4)$.

Remark 5.2 The axiomatisation algorithm from [2] requires the use auxiliary copying operators for non-smooth operators that have no implicit copies and test some of their arguments both positively and negatively (Lemma 5.2 in [2]). The examples of such operators are θ from the Introduction and the timed version of parallel operator of TPL in Example 2.8. Since we use orderings on rules instead of negative premises, our algorithm does not need to use auxiliary copying operators and rewrite rules for the mentioned type of operators: for example, we do not need the auxiliary operator $\triangle$ to deal with θ. So for these types of operators our method produces fewer auxiliary operators and rewrite rules than the method in [2].

5.2 Operators with no implicit copies and not simply distinctive

If an operator (f,n) is free of implicit copies and not simply distinctive, then $rules(f)$ and the ordering can be partitioned into a number of sets of simply distinctive rules that are unordered among themselves. The rules from different sets may be ordered. Such sets define auxiliary (simply distinctive) operators and we shall have a rewrite rule corresponding to the distinctifying law in [2]. Firstly, we need the following notation.

Definition 5.3 Let G be an OSOS PL with signature Σ that contains operators $+$ and $\rhd$. *Auxiliary form* of terms over G is defined using the notion of *sum terms* as follows:

(1) $f(\boldsymbol{X})$ is a sum term for each $f \in \Sigma \setminus \{+, \rhd\}$; if s and t are sum terms, then $t+s$ is a sum term.
(2) If s and t are sum terms, then $s \rhd t$ is in the auxiliary form; if s is a sum term and t is a term in the auxiliary form, then $s+t, t+s$ and $s \rhd t$ are terms in the auxiliary form.

Note that if s is a sum term and t is an auxiliary term, then $t \rhd s$ is not necessarily in auxiliary form, as is witnessed by $(f \rhd f' + f'') \rhd g$. Terms in the auxiliary form will be called auxiliary terms.

Lemma 5.4 *Let G be an OSOS PL with signature Σ such that $\mathsf{B} \leq G$. Let $\mathcal{P} = (\Sigma, \mathrm{R}, \succ)$ be a well-defined PRS for G that is sound for bisimulation. Suppose $(f,n) \in \Sigma$ is an operator with no implicit copies which is also not simply distinctive. Then, there is*

- *a disjoint extension G' of G with l simply distinctive operators (f_i,n), thus creating a new signature Σ',*
- *an auxiliary term $AuxiliaryTerm[f_1(\boldsymbol{X}), \ldots, f_l(\boldsymbol{X})]$ built from all operators (f_i,n), $+$ and $\rhd$ and involving only those operators, and*
- *a PRS $\mathcal{P}' = (\Sigma', \mathrm{R} \cup \{f_{aux}\}, \succ)$ with the new* **auxiliary** *rewrite rule below which*

is "unordered" with respect to the rewrite rules in R

$$f_{aux}: \ f(\boldsymbol{X}) \ \rightarrow \ \mathit{AuxiliaryTerm}[f_1(\boldsymbol{X}),\ldots,f_l(\boldsymbol{X})],$$

and the PRS $\mathcal{P}'$ is sound for bisimulation.

PROOF. We describe procedures to find the required distinctive operators and the auxiliary term, respectively, and then we show the soundness of the auxiliary rewrite rule. The details are given in Appendix A. □

It is clear from the proof that when the ordering on rules for f is empty, then the form of the auxiliary term is simply a sum:

Corollary 5.5 Let G, $\mathcal{P}$ and (f,n) be as in Lemma 5.4. If the ordering on rules for f is empty, then $\mathit{AuxiliaryTerm}[f_1(\boldsymbol{X}),\ldots,f_l(\boldsymbol{X})] \equiv \sum_{i=1}^{l} f_l(\boldsymbol{X})$.

The rest of this subsection is devoted to examples that illustrate the application of the procedures for the derivation of the auxiliary term and auxiliary rewrite rule.

Example 5.6 Let B be extended with "$\|$" the parallel composition operator of CCS. The operator is not simply distinctive but free of implicit copies. Assume that, for each $a \in \mathsf{Act}$, we have $\overline{a} \in \mathsf{Act}$ and $\overline{\overline{a}} = a$. Following the Auxiliary Term Generation Procedure we partition the rules for $\|$ into three sets: rules for the first argument, rules for the second argument and the communication rules. The resulting auxiliary operators are the left-merge, written as "$\between$", the right-merge, written as "$\between$", and the communication merge, written as "$|$", as in [10,2]. The defining rule schemas for these operators, for all $a \in \mathsf{Act}$, are as follows:

$$\frac{X \xrightarrow{a} X'}{X \between Y \xrightarrow{a} X' \| Y} \qquad \frac{Y \xrightarrow{a} Y'}{X \between Y \xrightarrow{a} X \| Y'} \qquad \frac{X \xrightarrow{a} X' \quad Y \xrightarrow{\overline{a}} Y'}{X \mid Y \xrightarrow{\tau} X' \| Y'}$$

We assume that prefixing binds stronger than the above three operators, and they in turn bind stronger than $+$ and $\rhd$. Since there is no ordering on the original rules for $\|$ there is no ordering between the rules for the three auxiliary operators. The initial and the final set S is $\{(\emptyset,\{\between\}),\ (\emptyset,\{\between\}),\ (\emptyset,\{|\})\}$ (i.e. the iteration routine does not alter S): see Appendix A. There is a single equation of the form (A.1), namely $AT \ = \ \widehat{\{\between\}} + \widehat{\{\between\}} + \widehat{\{|\}}$. Moreover, there are three equations of the form (A.3): $\widehat{\{\between\}} = X \between Y$, $\widehat{\{\between\}} = X \between Y$, and $\widehat{\{|\}} = X \mid Y$. Replacing the constants with their definitions, we obtain the auxiliary term $X \between Y \ + \ X \between Y \ + \ X \mid Y$ and the auxiliary rewrite rule:

$$X \| Y \ \rightarrow \ X \between Y \ + \ X \between Y \ + \ X \mid Y$$

Since there is no ordering on the rules the auxiliary term does not involve $\rhd$, and the auxiliary rewrite rule is an instance of the distinctifying law and rewrite rule in [2,14].

Example 5.7 The sequential composition operator form Example 2.7 is not simply distinctive. It is, however, free of implicit copies. Its rules can be partitioned into the rule for the first argument, r_{a*}, and the rules for the second argument, r_{*b}. We notice that the rules r_{a*} are above the rules r_{*b} for all a and b. The resulting simply distinctive auxiliary operators "$;_1$" and "$;_2$", required by Lemma 5.4, are defined by these two sets of rule schemas:

$$\frac{X \xrightarrow{a} X'}{X;_1 Y \xrightarrow{a} X';Y} \qquad \frac{Y \xrightarrow{b} Y'}{X;_2 Y \xrightarrow{b} Y'}$$

The initial and the final set S is $\{(\emptyset,\{;_1\}),\ (\{;_1\},\{;_2\})\}$. There is a single equation of the form (A.1), namely $AT = (X;_1 Y) \rhd \widehat{\{;_2\}}$ and there is one equation of the form (A.3): $\widehat{\{;_2\}} = X;_2 Y$. Replacing the constants with their definitions, we obtain the auxiliary term $(X;_1 Y) \rhd (X;_2 Y)$ and the auxiliary rewrite rule:

$$X;Y \ \rightarrow\ (X;_1 Y)\ \rhd\ (X;_2 Y)$$

Example 5.8 Consider a version of the CCS parallel that gives priority to communication over concurrency. The operator is defined simply by putting each and every communication rule for $\|$ above all the concurrency rules for both arguments of $\|$. As noted in Example 2.8 the GSOS definition of this operator is awkward. As in Example 5.6 we need the three auxiliary operators $\between$, $\between$ and $|$. The set S is $\{(\emptyset,\{|\}),\ (\{|\},\{\between\}),\ (\{|\},\{\between\})\}$. Following our procedure, S gets partitioned into two sets and the resulting two equations of the form (A.1) are $AT = \widehat{\{|\}}$ and $\widehat{\{|\}} = (X \mid Y) \rhd (\widehat{\{\between\}} + \widehat{\{\between\}})$. Also, as in Example 5.6, there are equations for $\widehat{\{\between\}}$ and $\widehat{\{\between\}}$. The resulting auxiliary rewrite rule is as follows:

$$X \,\|\, Y \ \rightarrow\ X \mid Y\ \rhd\ (X \between Y\ +\ X \between Y)$$

5.3 Simply distinctive operators

So far we have given rewrite rules for operators which are not free of implicit copies (Lemma 5.1) and rewrite rules for operators (and auxiliary operators) which are free of implicit copies but not simply distinctive (Lemma 5.4). Now we consider simply distinctive operators. We shall define several types of rewrite rules, namely distributivity, action and deadlock rewrite rules. First, we introduce some useful notation. When r has no implicit copies in the premises, the *trigger* of r is the n-tuple $(\lambda_1,\ldots,\lambda_n)$, where $\lambda_i = a_i$ if $i \in I$, and $\lambda_i = *$ otherwise. We often write λ

for $(\lambda_1,\ldots,\lambda_n)$, and $\lambda_I.\boldsymbol{X}$ denotes the vector $\lambda_1.X_1,\ldots,\lambda_n.X_n$ where if $\lambda_i = *$, then $\lambda_i.X_i$ is simply X_i.

Lemma 5.9 *Let G be an OSOS PL with Σ such that $\mathsf{B} \leq G$ and all operators in $\Sigma \setminus \Sigma_{\mathsf{B}}$ are free of implicit copies and simply distinctive. Suppose $(f,n) \in \Sigma \setminus \Sigma_{\mathsf{B}}$ has the defining rules of the following form, where $Y_i = X_i'$ if $i \in I$, and $Y_i = X_i$ otherwise:*

$$\frac{\{\, X_i \xrightarrow{a_i} X_i' \,\}_{i\in I}}{f(X_1,\ldots,X_n) \xrightarrow{a} C[\boldsymbol{Y}]} \tag{3}$$

(1) For each active argument i of f the following are the **distributivity** *rewrite rules for f and i:*

$$f_{dn(i)} : f(\ldots,X_i+\mathbf{0},\ldots) \to f(\ldots,X_i,\ldots)$$
$$f_{ds(i)} : f(\ldots,X_i+Y_i,\ldots) \to f(\ldots,X_i,\ldots)+f(\ldots,Y_i,\ldots)$$

The priority order is $f_{dn(i)} \succ f_{ds(i)}$ for each $i \in I$.

(2) For each rule of the form (3) with $I \neq \emptyset$ and trigger $\boldsymbol{a}_I.\boldsymbol{X}$ the **action** *rewrite rule has the form:*

$$f^a_{act} : f(\boldsymbol{a}_I.\boldsymbol{X}) \to a.C[\boldsymbol{X}]$$

If f has no active arguments, then f^a_{act} is $f(\boldsymbol{X}) \to a.C[\boldsymbol{X}]$.

(3) The **deadlock** *rewrite rule is as follows:*

$$f_{nil} : f(\boldsymbol{X}) \to \mathbf{0}$$

The priority order satisfies $\{f_{ds(i)}, f^a_{act}\} \succ f_{nil}$ for all $f_{ds(i)}$ and f^a_{act}.

Let $\mathcal{P} = (\Sigma, \mathrm{R}, \succ')$ be $\mathcal{B}$, the PRS for B as in Figure 1, extended with all the distributivity, action and deadlock rewrite rules for each operator f as above, and let $\succ'$ be $\succ$ as in Figure 1 extended with the orderings required for the added rewrite rules. Then, $\mathcal{P}$ is sound for bisimulation and head normalising for all closed terms over Σ.

PROOF. See Appendix B.

Note, that soundness of $f_{ds(i)}$ rewrite rules does not depend on them being below the corresponding $f_{dn(i)}$ rewrite rules. Similarly, soundness of the deadlock rewrite rules does not depend on them being below the $f_{dn(i)}$ rewrite rules. This can be seen in the above proof. The distributivity rewrite rules $f_{dn(i)}$ are included purely for the purpose of resolving some of the inherent nondeterminism that is present in rewriting. More specifically, the inclusion of the rules $f_{dn(i)}$ resolves a large proportion of

this nondeterminism and, as a result, makes the task of proving confluence easier: see a proof of Theorem 7.1.

Also, note that if f is simply distinctive and it has at least two rules, then the premises of all rules for f are not empty. And, if f is simply distinctive and it has a defining rule with no premises, then this rule is its sole defining rule and f will have only the action rewrite rule and the deadlock rewrite rule.

5.4 Operators with one argument

There are free of implicit copies and not simply distinctive operators which have simpler rewrite rules than the auxiliary rewrite rules introduced in the previous subsection. These rules are called *priority resolving* rewrite rules. In this subsection we define a class of such operators: they must have a single argument and be simply distinctive when the ordering on their rules is removed. This class contains for instance the mentioned priority operator and the the hiding operator of ET-LOTOS [24]: see Example 5.11.

Lemma 5.10 *Let G be an OSOS PL with signature Σ such that $\mathsf{B} \leq G$. Let $\mathcal{P} = (\Sigma, \mathrm{R}, \succ)$ be a well-defined PRS for G that is sound for bisimulation. Suppose $(f,1) \in \Sigma$ is a free of implicit copies operator which is not simply distinctive, and the ordering on rules for f is not empty. Moreover, let f be such that it is simply distinctive when we remove the ordering on its rules. Suppose the rules for f have the following form:*

$$\frac{X \xrightarrow{a} X'}{f(X) \xrightarrow{\alpha} C[X']} \tag{4}$$

For each pair of distinct rules r and r' of the form (4) such that $r > r'$, and for triggers $a.Y$ and $b.Z$ of r and r', respectively, the **priority resolving** *rewrite rule for the rule r is as follows:*

$$f^{r}_{pr} : f(X + a.Y + b.Z) \rightarrow f(X + a.Y)$$

Also, let the new version of f be without the ordering on its rules, so the new f is simply distinctive. Then, there is a PRS $\mathcal{P}' = (\Sigma, \mathrm{R}', \succ')$, where R' is R extended with all priority resolving rewrite rules for the original f as required above, and all rewrite rules for the new f as required by Lemma 5.9. The ordering $\succ'$ is $\succ$ extended by putting every priority rewrite rule f^{r}_{pr} above the rule $f_{dn(1)}$ as in Lemma 5.9, and by adding the orderings introduced by Lemma 5.9. Then the PRS $\mathcal{P}'$ is sound for bisimulation.

PROOF. In the last section we showed the soundness of the rewrite rules required by Lemma 5.9. Hence, it remains to prove the soundness of the priority resolving rewrite rules for operators f as in the lemma. Since the rules for f with the ordering removed define a simply distinctive operator there is at most one rule with the premise $X \xrightarrow{a} X'$ for every action a. Also, by the definition of the orderings on SOS rules, if $r > r'$ then $r' > r$ is false for any two rules r and r' for any f.

Let r and r' be of the form (4) with the triggers $a.X$ and $b.X$, respectively, and let $r > r'$. It is enough to show $f(p+a.q+b.r) \xrightarrow{\alpha} t$ iff $f(p+a.q) \xrightarrow{\alpha} t$. Assume $f(p+a.q+b.r) \xrightarrow{\alpha} t$. This transition implied two cases: either the rule r is enabled with the trigger $a.q$ or there is a rule r'' not below r and enabled with the trigger $c.p'$, and $p \xrightarrow{c} p'$. In the first case, since r is enabled it means no rule in $higher(r)$ is enabled with the argument $p+a.q+b.r$. Since r fires with $a.q$ it also fires with $p+a.q$, so, $f(p+a.q) \xrightarrow{\alpha} t$ as required. In the second case no rule in $higher(r')$ is enabled with the argument $p+a.q+b.r$. Hence, no rule in $higher(r')$ is enabled with simpler $p+a.q$ and, consequently, $f(p+a.q) \xrightarrow{\alpha} t$. The converse also follows by a similar argument. □

The priority operator θ is the only operator discussed so far that can be dealt with by Lemma 5.10. It has one argument, non-empty ordering on the rules and it becomes simply distinctive when the ordering on the rules is removed. All the priority rewrite rules for θ required by Lemma 5.10 have been given in the Introduction.

Another operator that can be dealt with by Lemma 5.10 is the hiding operator of ET-LOTOS [24]:

Example 5.11 Our definition of the hiding operator **hide** employs an ordering on the defining rules instead of negative premises and a *lookahead* as in [24]. The two traditional rules for the operator, where the second rule is denoted r_a for each $a \in A$, are:

$$\frac{X \xrightarrow{a} X'}{\textbf{hide } A \textbf{ in } X \xrightarrow{a} \textbf{hide } A \textbf{ in } X'} \; a \notin A \qquad \frac{X \xrightarrow{a} X'}{\textbf{hide } A \textbf{ in } X \xrightarrow{\tau} \textbf{hide } A \textbf{ in } X'} \; a \in A$$

The required timed rule r_σ, where σ denotes the passage of one time unit, is simply

$$\frac{X \xrightarrow{\sigma} X'}{\textbf{hide } A \textbf{ in } X \xrightarrow{\sigma} \textbf{hide } A \textbf{ in } X'} \; r_\sigma$$

and the ordering is $r_\sigma < r_a$ for all $a \in A$. Clearly, the operator satisfies the requirements of Lemma 5.10. The priority resolving rewrite rules are given below, one for

every a in A:

$$\mathbf{hide}^{a}_{pr} : \mathbf{hide}\ A\ \mathbf{in}\ (a.X + \sigma.Y + Z) \to \mathbf{hide}\ A\ \mathbf{in}\ (a.X + Z)$$

Moreover, there are other rewrite rules required by Lemma 5.10. We obtain them by removing the ordering on the rules for **hide** and then applying Lemma 5.9:

$$\begin{aligned}
\mathbf{hide}_{dn} :&\ \ \mathbf{hide}\ A\ \mathbf{in}\ (X + \mathbf{0}) \to \mathbf{hide}\ A\ \mathbf{in}\ (X)\\
\mathbf{hide}_{ds} :&\ \ \mathbf{hide}\ A\ \mathbf{in}\ (X + Y) \to \mathbf{hide}\ A\ \mathbf{in}\ X\ +\ \mathbf{hide}\ A\ \mathbf{in}\ Y\\
\mathbf{hide}^{a}_{act} :&\quad \mathbf{hide}\ A\ \mathbf{in}\ (a.X) \to a.(\mathbf{hide}\ A\ \mathbf{in}\ X)\\
\mathbf{hide}^{\tau}_{act} :&\quad \mathbf{hide}\ A\ \mathbf{in}\ (a.X) \to \tau.(\mathbf{hide}\ A\ \mathbf{in}\ X)\\
\mathbf{hide}^{\sigma}_{act} :&\quad \mathbf{hide}\ A\ \mathbf{in}\ (\sigma.X) \to \sigma.(\mathbf{hide}\ A\ \mathbf{in}\ X)\\
\mathbf{hide}_{nil} :&\qquad \mathbf{hide}\ A\ \mathbf{in}\ X \to \mathbf{0}
\end{aligned}$$

Here, we have one $\mathbf{hide}^{a}_{act}$ rule for every $a \notin A \cup \{\sigma\}$, and one $\mathbf{hide}^{\tau}_{act}$ for every $a \in A$. The priority order $\succ$ satisfies $\mathbf{hide}^{a}_{pr} \succ \mathbf{hide}_{dn}$ for all priority resolving rules $\mathbf{hide}^{a}_{pr}$ as required by Lemma 5.10. Moreover, Lemma 5.9 requires $\mathbf{hide}_{dn} \succ \mathbf{hide}_{ds}$, and $\{\mathbf{hide}_{ds}, \mathbf{hide}^{a}_{act}, \mathbf{hide}^{\tau}_{act}, \mathbf{hide}^{\sigma}_{act}\} \succ \mathbf{hide}_{nil}$.

5.5 The PRS algorithm, head normalisation and soundness

In the previous subsections we defined priority rewrite rules for several classes of OSOS operators and proved that they are sound for bisimulation. Presently, we show that PRSs generated by our approach for PLs that extend B and contain no operators with implicit copies are head normalising. This is a consequence of Lemmas 5.4, 5.10 and 5.9: see Appendix C.

Lemma 5.12 *Let G be an OSOS PL with signature Σ such that $\mathsf{B} \leq G$, and let all operators in $\Sigma \setminus \Sigma_{\mathsf{B}}$ be free of implicit copies. Then, there is disjoint extension G' of G with a finite collection of $\Sigma_{G'} \setminus \Sigma$ simply distinctive operators, and a PRS $\mathcal{P}$ that contains the PRS for B and is sound for bisimulation and head normalising.*

The rest of the subsection presents the algorithm in Figure 2 for generating PRSs for arbitrary OSOS PLs. We also prove head normalisation and soundness for bisimulation for the generated PRSs.

Theorem 5.13 *Let G be an OSOS process language, and let G' and $\mathcal{P}$ be the OSOS process language and the PRS respectively that are produced by the algorithm in Figure 2. Then, $\mathcal{P}$ is head normalising and sound for bisimulation.*

Input: OSOS process language $G = (\Sigma_G, A, R, >)$ and PRS $\mathcal{P} = (\Sigma_G, \emptyset, \emptyset)$.

(1) If G is not a disjoint extension of B then add to G a disjoint copy of B. Call the resulting language G'''. $\mathcal{P}$ becomes $(\Sigma_{G'''}, \mathrm{R}'', \succ')$, where R'' and $\succ'$ are rewrite rules and priority order for B as in Figure 1.

(2) For each operator $f \in G'''$ which is not free of implicit copies apply the construction of Lemma 5.1 to obtain a free of implicit copies operator f^c. G''' extended disjointly with all f^c, for all not free of implicit copies operators f of G''', is denoted by G''. $\mathcal{P}$ becomes $(\Sigma_{G''}, \mathrm{R}''_{copy}, \succ')$, where R''_{copy} is R'' extended with all the copying rewrite rules required by Lemma 5.1.

(3) For each free of implicit copies operator $f \in \Sigma_{G''} \setminus \Sigma_{\mathsf{B}}$ which is not simply distinctive, and which **satisfies** the conditions of Lemma 5.10, extend R''_{copy} with all the priority resolving rewrite rules and as in Lemma 5.10 to obtain the new PRS $\mathcal{P} = (\Sigma_{G'}, \mathrm{R}''_{pr}, \succ')$. The PL G' is the result of extending G'' disjointly with the (simply distinctive) versions of all such operators.

(4) For each free of implicit copies operator $f \in \Sigma_{G''} \setminus \Sigma_{\mathsf{B}}$ which is not simply distinctive, and which **does not satisfy** the conditions of Lemma 5.10, apply the construction of Lemma 5.4 to produce simply distinctive auxiliary operators $f_1, \ldots, f_l$. The PL G' is the result of extending G'' disjointly with all auxiliary operators for all such operators. $\mathcal{P}$ becomes $(\Sigma_{G'}, \mathrm{R}', \succ')$, where R' is R''_{pr} extended with all the auxiliary rewrite rules required by Lemma 5.4.

(5) For each simply distinctive operator f in $\Sigma_{G'} \setminus \Sigma_{\mathsf{B}}$ extend R' and $\succ'$ with all the distributivity, action and deadlock rewrite rules and the associated priority orders as in Lemma 5.9 and, if necessary, in Lemma 5.10. The resulting PRS $\mathcal{P}$ is $(\Sigma_{G'}, \mathrm{R}, \succ)$.

Output: OSOS PL G' such that $G \leq G'$, and a sound for bisimulation and head normalising PRS $\mathcal{P}$.

Fig. 2. The PRS algorithm for OSOS process languages.

PROOF. Given a PL G, the algorithm firstly extends G disjointly with B producing G''', and the PRS in Figure 1 becomes the basis for the required $\mathcal{P}$. Then, it considers each of the operators of G''' in turn and generates rewrite rules with priorities as described in the previous subsections, and accumulates them into the required $\mathcal{P}$.

If f is an operator of G''' with implicit copies, then by Lemma 5.1 we can extend G''' disjointly with a free of implicit copies operator f^c. We add the copying rewrite rule to the current PRS. We carry out this procedure, step (2) of the algorithm, for all operators of G''' with implicit copies. It produces a PL G'', and the constructed so far PRS has all the required copying rewrite rules.

Next, we consider operators of G'' which are free of implicit copies but which are not simply distinctive and apply the strategy described in Section 5.2. There are two routes that the algorithm can take at this point, namely steps (3) and (4). For all such operators that fail the conditions of Lemma 5.10 we apply Lemma 5.4 and add auxiliary rewrite rules: step (4) of the algorithm. But for operators which satisfy the conditions of Lemma 5.10, such as the priority operator (Introduction) and the hiding operator (Example 5.11), we apply the strategy of Section 5.4 and add the priority resolving rewrite rules as in Lemma 5.10: step (3). After steps (3) and (4) have been applied to all appropriate operators, we obtain a PL G'. The enlarged PRS contains at this point all the auxiliary rewrite rules and the priority resolving rewrite rules.

Finally, we perform step (5): for each simply distinctive operator in G' we add to the current PRS all the distributivity, action and deadlock rewrite rules and the associated priority orders as in Lemma 5.9 and, if necessary, as in Lemma 5.10. Thus we obtain the required PRS $\mathcal{P}$, which is sound for bisimulation. It is also head normalising for closed terms over G' which are built from free of implicit copies operators and the operators of B only: see Lemma 5.12. The remaining operators of G', namely operators with implicit copies, give rise to copying rewrite rules as in Lemma 5.1. Since terms on the right hand side of such rules have free of implicit copies operators as the outermost operators, we see that terms constructed with operators with implicit copies rewrite to hnf. □

6 Termination

Any practically useful PL must contain a mechanism for representing processes with infinite behaviour. Most often this is done by means of process constants (or variables) that are defined by *mutual recursion*. For example, a unary semaphore can be represented by a process with two states *Sem* and *Sem*′ defined by $Sem \xrightarrow{up} Sem'$ and $Sem' \xrightarrow{down} Sem$ respectively. *Sem* and *Sem*′ are simply distinctive, free of implicit copies OSOS operators. By Lemma 5.9, the only priority rewrite rules for these operators are the following action and the deadlock rules:

$$Sem \to up.Sem' \succ Sem \to \mathbf{0} \qquad \text{and} \qquad Sem' \to down.Sem \succ Sem' \to \mathbf{0}$$

It is not surprising that processes such as *Sem* have non-terminating reductions:

$$Sem \to up.Sem' \to up.down.Sem \to up.down.up.Sem' \to \cdots$$

The properties of PRSs with operators such as *Sem* are the subject of *infinitary rewriting* [21]. However, there is an interesting subclass of OSOS PLs that contain

only those operators that lead to finite behaviours. The PRSs generated by algorithm in Figure 2 for PLs in this subclass will be strongly normalising (terminating) for closed terms modulo *AC* of $+$.

We define PLs and processes with finite behaviour. Following [2] we have:

Definition 6.1 Let G be an OSOS process language. 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 OSOS PLs is not decidable, but syntactical well-foundedness is decidable by the corresponding argument as in [2]. Moreover, if a PL is linear as well as syntactically well-founded, then it is well-founded [2]. These two new notions are defined again following [2]:

Definition 6.2 An OSOS transition rule of the form (2) 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. An OSOS operator is linear if all its transition rules are linear. An OSOS PL is linear if all its operators are linear.

Definition 6.3 An OSOS PL G is *syntactically well-founded* 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 $W(f(t_1, \ldots, t_n)) \stackrel{def}{=} w(f) + W(t_1) + \cdots + W(t_n)$.

It can be easily shown by solving a linear system of Diophantine equations that syntactical well-foundedness of OSOS PLs is decidable, and if an OSOS PL is linear as well as syntactically well-founded, then it is also well-founded [2]. Most of the commonly used process operators, and all operators mentioned in this paper, are linear. As for syntactical well-foundedness, any PL with constants defined by mutual recursion does not satisfy it: since $Sem \xrightarrow{up} Sem'$ and $Sem' \xrightarrow{down} Sem$, there is no w such that $w(Sem) > W(Sem') = w(Sem')$ and $w(Sem') > W(Sem) = w(Sem)$. Apart from recursively defined process constants, the basic PL B extended with any operators described in the paper, and with many more operators from standard PLs, is syntactically well-founded. Typically, we assign weight 1 to the action prefixing operators and weight 0 to other operators. Further discussion related to this topic is in Appendix D.

Theorem 6.4 *Let G be a syntactically well-founded and linear OSOS process language, and let G' and $\mathcal{P}$ be the OSOS process language and the PRS respectively that are produced by the algorithm in Figure 2. Then, $\mathcal{P}$ is strongly normalising modulo AC on closed terms over G'.*

PROOF. The details are given in Appendix D. There, we employ novel techniques of dependency pairs and dependency graphs adapted to rewriting modulo *AC* of the choice operator $+$. $\Box$

7 Confluence and Completeness for Bisimulation

The algorithm in Figure 2 produces, for any OSOS process language G, a disjoint extension G' and a PRS for G'. If G is syntactically well-founded and linear, then the PRS for G' is strongly normalising. We show that if the PRS for G' is strongly normalising, then it is also confluent. We shall use the classical result due to Knuth and Bendix [22] that states that if a TRS is strongly normalising, then it is confluent if and only if all its critical pairs are convergent. The main purpose of priority orders is to resolve the ambiguity concerning the choice of overlapping rules when rewriting terms. The priority order produced by the algorithm in Figure 2 resolves a large proportion of this ambiguity by reducing the number of critical pairs, and thus making the task of proving confluence a lot easier. We also have completeness result for bisimulation:

Theorem 7.1 *Let G be a syntactically well-founded and linear OSOS process language, and let G' and $\mathcal{P}$ be the OSOS process language and the PRS respectively that are produced by the algorithm in Figure 2. Then, $\mathcal{P}$ is confluent and complete for bisimulation on closed terms over G'.*

PROOF. There are several critical pairs for the $\mathcal{B}$ component of $\mathcal{P}$ and we dealt with them in Theorem 4.3. We list the remaining overlapping rewrite rules and if the priority order permits ambiguous reductions, we list the resulting critical pairs and show that they are joinable. Due to the form of our rewrite rules and the priority order on the rules there are only a few simple types of critical pairs. Case (2) explains the reason for having distributivity rewrite rules $f_{dn}(i)$.

(1) The rewrite rule $X+\mathbf{0}\rightarrow X$ overlaps with the distributivity rules $f_{dn}(i)$ for all simply distinctive operators f and all their active arguments i. The resulting critical pairs $(f(\ldots,X_i,\ldots), f(\ldots,X_i,\ldots))$ are joinable.
(2) The rewrite rule $X+\mathbf{0}\rightarrow X$ overlaps with the distributivity rules $f_{ds(i)}$ for all simply distinctive operators f and all their active arguments i. However, because $f_{dn(i)} \succ f_{ds(i)}$ and since $f_{dn(i)}$ is applicable to $f(\ldots,X+\mathbf{0},\ldots)$, there are no critical pairs for such overlaps.
(3) The rule $X+X\rightarrow X$ overlaps with all the relevant distributivity rules $f_{dn}(i)$ and $f_{ds}(i)$ for all simply distinctive operators f and all their active arguments i. The resulting critical pairs are joinable.
(4) The rule $X+\mathbf{0}\rightarrow X$ can overlap with the priority resolving rule $f(X+a.Y+b.Z)\rightarrow f(X+a.Y)$. The resulting critical pair is $(f(X+a.Y+b.Z),\ f(X+\mathbf{0}+$

$a.Y$). Then, the first element of the pair reduces by f^a_{pr} to $f(X + a.Y)$, and the second element of the pair reduces by $+_{dn}$ to the same $f(X + a.Y)$.

Finally, we consider completeness for bisimulation. Since $\mathcal{P}$ is strongly normalising it is well-defined. Since $\mathcal{P}$ is confluent each closed G' term can be reduced to unique normal form. As $\mathcal{P}$ is head normalising, and G' is well-founded, we can show by structural induction that each closed G' term can be reduced to a unique B term in nf. Since $\mathcal{P}$ is sound for bisimulation, it is now sufficient to prove that the PRS for B is complete for bisimulation. Indeed, this is Theorem 4.4. □

8 Conclusion and Possible Extensions

We have described how to produce, for an arbitrary OSOS PL, a PRS that is head normalising and sound for bisimulation. When a PL in question is syntactically well-founded and linear, then its PRS is strongly normalising and confluent, and two processes are bisimilar if and only if they can be reduced to the same normal form modulo *AC*. We believe that our procedure can be adapted to other classes of PLs and other process equivalences such as, for example, a subclass of De Simone PLs and testing equivalence [28,38].

In the concurrency literature there are well developed techniques for equational reasoning for non-well-founded processes. For example, consider *regular* processes [26] and reasoning about such processes with respect to bisimulation. One can prove equalities between such processes by using the standard axioms to "unwind" guarded recursive processes to head normal form, and the *Recursive Specification Principle* (RSP) [10]. It would be worth investigating how a class of infinitary OSOS PLs corresponding to Aceto's class of *regular infinitary* GSOS PLs [1] can be given a rewrite system that is sound and complete for bisimulation. Such rewrite system would contain rewrite versions of the Recursive Specification Principle rules.

It would be interesting to investigate further the benefits of priority orderings on rewrite rules. Apart from reducing nondeterminism inherent in rewriting, could they be also used to internalise rewrite strategies thus improving weak normalisation to strong normalisation?

Acknowledgements

We would like to thank the referees for their comments and suggestions. The first author would like to thank the University of Leicester for granting study leave, and acknowledge gratefully support from EPSRC, grant EP/D001307/1, and from

Nagoya University during a research visit. The second author would like to thank the Kayamori Foundation for Information Science Advancement for supporting a visit to the University of Leicester. Thanks are also due to Paul Taylor for his Proof Trees and Commutative Diagrams macros.

Appendix

A Proof of Lemma 5.4

We find the auxiliary operators f_i by partitioning $\mathit{rules}(f)$ into sets such that the operators defined by the rules in each of the sets are simply distinctive, and the resulting sets satisfy the *ordering condition* that we define below. We shall need new binary relations $\ggg$ and $\gg$ on sets of rules: $R \ggg R'$ if for all $r' \in R'$ and $r \in R$ we have $r > r'$, and $R \gg R'$ if for all $r' \in R'$ there is $r \in R$ such that $r > r'$. Clearly, $\ggg \subseteq \gg$ but not $\gg \subseteq \ggg$. The relations $\ggg, \gg$ are irreflexive and transitive. We shall write $R_i \nmid R_j$ if the ordering between the rules in R_i and R_j is empty.

The *initial partition* is achieved as follows. Let AX be the set of all axioms in $\mathit{rules}(f)$, namely rules with no premises. If AX is non-empty, then partition AX into singleton sets. Then, partition $\mathit{rules}(f) \setminus AX$ into sets as large as possible in such way that each set consists of rules with premises for the same arguments, and no two different sets have rules with the same arguments in the premises.

The following condition shall be useful:

Definition A.1 The set $\{R_1, \ldots, R_n\}$ satisfies the *ordering condition* if for every two member sets R_i and R_j either $R_i \ggg R_j$, $R_j \ggg R_i$ or $R_i \nmid R_j$.

If the partition obtained so far does not satisfy the ordering condition, for example because a rule in R_1 is not above all rules in R_2, then split further the offending partition sets into as large as possible subsets until the ordering condition holds in the resulting partition. This gives us the *final partition* $R_1, \ldots, R_l$ of $\mathit{rules}(f)$. Note, that some R_is may have rules that are ordered among themselves (as for the priority operator θ which is simply distinctive). In each R_i we change the operator in the source of each rule from f to f_i thus obtaining R'_i. So, we have constructed l simply distinctive n-ary auxiliary operators $f_1, \ldots, f_l$ and their defining rules $R'_1, \ldots, R'_l$, respectively.

Next, we present *Auxiliary Term Generation Procedure* for deriving the auxiliary term given the simply distinctive operators $f_1, \ldots, f_l$ and their rules $R'_1, \ldots, R'_l$. The procedure consists of four steps. We shall require more notation: We write $f_i \ggg f_j$ if $R'_i \ggg R'_j$, and $f_i \gg f_j$ if $R'_i \gg R'_j$. The ordering $\ggg$ and $\gg$ are extended to sets of

repeat
$S := S'$
for each $(F_i', F_i) \in S$ **do**
$F := \emptyset$
if $F_i' = \emptyset$
then $S' := S' \cup \{(F, F_i)\}$
else for each $(F_j', F_j) \in S$ **do**
if $F_j \subseteq F_i'$
then $F := F \cup F_j'$
od
$S' := S' \cup \{(F, F_i')\}$
od
until $S' = S$

Fig. A.1. "Upwards closure" iteration routine

operators in the standard way. We say that operator f_i is "fully above" f_j if $f_i \ggg f_j$ and there is no other operator f_k such that $f_i \ggg f_k \ggg f_j$. Given f_j, $\mathit{fabove}(f_j)$ is the set of all operators fully above f_j. The function *fabove* generalises to sets of operators in the standard way. Additionally, we shall use the set of operators "above" F, where F is itself a set of operators, written as $\mathit{above}(F)$. We define $\mathit{above}(F)$ as $\bigcup_{f \in F} \mathit{fabove}(f)$. As a result we have $\mathit{above}(F) \gg F$ but not necessarily $\mathit{above}(F) \ggg F$.

Auxiliary Term Generation Procedure

Input: Simply distinctive operators $f_1, \ldots, f_l$ and their rules $R_1', \ldots, R_l'$, respectively.

Step 1. We calculate for each auxiliary operator the sets of auxiliary operators fully above it. This is done by constructing the (initial value of) set the S:

$$S = \{(\mathit{fabove}(f_1), \{f_1\}), \ldots, (\mathit{fabove}(f_l), \{f_l\})\}$$

Note that $\mathit{fabove}(f_i) \ggg \{f_i\}$ for all $1 \leq i \leq l$.

Step 2. We produce an "upward closure" of S with respect to the ordering $\gg$. We aim to enlarge S so that for each pair $(F, G) \in S$ with $F = \mathit{above}(G)$ there is a unique pair $(H, F) \in S$ such that $H = \mathit{above}(F)$. This closure is achieved by performing the following procedure, where F, F_i, F_i', F_j and F_j' are sets of auxiliary operators. First, we assign S to S' ($S' ::= S$). Then, we perform the iteration in Figure A.1. Since the initial set S is finite and the iteration enlarges S by adding pairs whose first and second components are subsets of the finite set $\{f_1, \ldots, f_l\}$, the iteration eventually terminates. Let the resulting set S be as follows, where $n \geq l$:

$$\{(F_1', F_1), \ldots, (F_n', F_n)\}$$

We suspend the description of the procedure in order to list several important properties of the resulting set S. They will be used in the proof of soundness of the auxiliary rewrite rule:

(1) $F_i' = above(F_i)$ for all $1 \leq i \leq n$.
(2) For all $1 \leq i \leq l$ the set S contains the pair $(G_i, \{f_i\})$ for some possibly empty G_i.
(3) For all $(F_i', F_i) \in S$ we have $F_i \neq \emptyset$. Also, there exists $K \subseteq \{1, \ldots, l\}$ such that $F_k' = \emptyset$ for all $k \in K$.
(4) If $F' \neq \emptyset$ and $(F', F) \in S$, then $(F'', F') \in S$ for some possibly empty F''.
(5) If $(F', F) \in S$, then either (a) there exists $(F, G) \in S$ for some G or (b) $(F, G) \notin S$ for all G, and then $F = \{f_k\}$ for some $k \in \{1, \ldots, l\}$,
(6) S may contain two pairs (F', F_i) and (F', F_j) such that $F_i \neq F_j$ and not(see Example 5.8) (different sets may have upper bounds), but never contains two pairs (F'', F) and (F', F) such that $F' \neq F''$ (different sets may not have lower bounds).

The above properties imply that the pairs of the form $(\emptyset, F_k)$ indicate that the set $\emptyset$ is maximal in the ordering generated by $\gg$, and that the pairs of the form $(G_i, \{f_i\})$ indicate that the sets $\{f_i\}$ are minimal in the ordering generated by $\gg$. Also, they imply the existence of upward chains of sets F_i ordered by $\gg$ with the bottom element $\{f_j\}$ and the top element $\emptyset$ for $1 \leq j \leq l$. In short, the set S defines an upside-down tree: $\emptyset$ is the root, the sets $\{f_i\}$ are the leaves, and for each F_i in such a structure, the set of F_js such that $F_j \gg F_i$ is a chain.

Now we return to our procedure. The last two steps (Step 3 and Step 4 below) describe how to construct the auxiliary term. Firstly, using the set S, we create a number of process constants and derive their defining equations. There will be two types of such defining equations. The right-hand sides of the equations contain at most (other) process constants, auxiliary operators f_i and the operators $+$ and $\triangleright$. Hence, we get a number of equations that define constants in terms of each other but not recursively. Once we have the set of such equations, we replace the constants on the right-hand sides of the equations by their definitions and thus obtain the required auxiliary term.

Step 3. We partition S into sets $S_1, \ldots, S_k$, for $1 \leq k \leq n$, such that all pairs of S with the same first element belong to precisely one partition. Each of the partitions gives rise to one constant and its defining equation. A typical partition has the form $\{(F_i', F_{i1}), \ldots, (F_i', F_{im_i})\}$ and it produces the constant $\widehat{F_i'}$, which is named after the first element of the pairs in the partition. Here, the construct "$\widehat{\ }$" is used to make process constants out of symbols F_i. This partition gives rise to the following defining equation:

$$\widehat{F_i'} \stackrel{\text{def}}{=} \sum_{l=1}^{m_i} \Big(\sum_{f_k \in F_i'} f_k(\boldsymbol{X}) \Big) \triangleright \widehat{F_{il}}$$

Since $X \rhd Y + X \rhd Z = X \rhd (Y+Z)$ holds for bisimulation we simplify the above equation accordingly, and obtain the first type of equations for the constants:

$$\widehat{F_i'} \stackrel{\text{def}}{=} (\sum_{f_k \in F_i'} f_k(\boldsymbol{X})) \rhd (\sum_{l=1}^{m_i} \widehat{F_{il}}) \tag{A.1}$$

When $F_i' = \emptyset$, then an equation of the type (A.1) becomes $\widehat{F_i}' = \mathbf{0} \rhd (\sum_{l=1}^{m_i} \widehat{F_{il}})$. As $\mathbf{0} \rhd X = X$ we obtain simply $\widehat{F_i}' = \sum_{l=1}^{m_i} \widehat{F_{il}}$. Since S contains always one or more pairs $(\emptyset, F_m)$, for some non-empty F_m, this equation plays a special role and we shall use a fresh constant AT instead of $\widehat{\emptyset}$ and write the equation as:

$$AT \stackrel{\text{def}}{=} \sum_{l=1}^{m_i} \widehat{F_{il}} \tag{A.2}$$

So far, we have created a constant $\widehat{F}$ for every $(F,G) \in S$. Additionally, we shall also need constants and defining equations for some of the sets G. These constants arise from sets G that are not above any other sets (although for some G' the set $G \cup G'$ may be above another set). More precisely, for each $(F,G) \in S$ such that $(G,H) \notin S$ for all H, the set G is a singleton set, say $\{f_g\}$, and we have the equation of the second type:

$$\widehat{G} = f_g(\boldsymbol{X}) \tag{A.3}$$

Hence, there is a constant $\widehat{F_i}$ for $F_i = \{f_i\}$ for all $1/leqi/leql$. And, if $\widehat{F}$ appears on the right-hand side of one of the equations above, then there is also a defining equation for that constant. This is a consequence of the way we constructed S and its resulting properties.

Step 4. We replace each constant that appears on the right-hand side of the equation for AT, (A.2), by the right-hand side of its defining equation. We repeat this until the term on the right-hand side has no constants. It can be seen easily that this process terminates successfully using the observations from the previous paragraphs. The obtained term (on the right-hand side) is in the auxiliary form. We denote this term by $AuxiliaryTerm[f_1(\boldsymbol{X}), \ldots, f_l(\boldsymbol{X})]$ as required in Lemma 5.4.

Output: $AuxiliaryTerm[f_1(\boldsymbol{X}), \ldots, f_l(\boldsymbol{X})]$.

Finally, we require a proof that the auxiliary rewrite rule is sound for bisimulation. It is sufficient to prove that $f(\boldsymbol{p}) \xrightarrow{a} t$ iff $AuxiliaryTerm[f_1(\boldsymbol{p}), \ldots, f_l(\boldsymbol{p})] \xrightarrow{a} t$ for some vector of terms $\boldsymbol{p}$ and term t over G, and some action a.

Only if part. Let $f(\boldsymbol{p}) \xrightarrow{a} t$ be derived by rule r with a ground substitution σ. Assume that $r \in R_k$ where $R_1, \ldots, R_l$ is the final partition of the rules for f. Also let r_k

be the rule r but with f in the source replaced by f_k. So, $r_k \in R'_k$ is the rule for the auxiliary f_k that corresponds to the rule r for f. Clearly, the targets of both rules are identical and, under σ, are equal to t. Hence, $f_k(\boldsymbol{p}) \xrightarrow{a} t$ is derivable by the rule r_k. It remains to be shown how $f_k(\boldsymbol{p}) \xrightarrow{a} t$ implies $\mathit{AuxiliaryTerm}[f_1(\boldsymbol{p}), \ldots, f_l(\boldsymbol{p})] \xrightarrow{a} t$.

The transition $f(\boldsymbol{p}) \xrightarrow{a} t$ implies that no rule in $\mathit{higher}(r)$ is enabled under substitution σ. We construct the corresponding set of rules higher than r_k among the rules in $R'_1, \ldots, R'_l$ as follows. We denote by $\mathit{Higher}(f_k)$ the set of rules higher than f_k in the ordering $\gg$ as given by the set S. Clearly, the set of rules higher than r_k is a subset of $\mathit{Higher}(f_k)$. By the construction of S there is a sequence $G_1, \ldots, G_m$ of sets of auxiliary operators above $\{f_k\}$; assume that $G_0 = \{f_k\}$. We have $G_1 = \mathit{above}(\{f_k\})$, $G_2 = \mathit{above}(G_1)$ and so on, with $G_m = \emptyset$. Hence, $\mathit{Higher}(f_k) = \bigcup_{j=1}^{m} \mathit{rules}(G_j)$. The construction of sets the G_i gives us the constants $(\widehat{\{f_k\}} =)\widehat{G_0}$ and $\widehat{G_1}, \ldots, \widehat{G_m}$, and the following equations (best read from bottom up):

$$
\begin{aligned}
\mathit{AuxiliaryTerm}[f_1(\boldsymbol{p}), \ldots, f_l(\boldsymbol{p})] &= \cdots + \widehat{G_m} + \cdots \\
&\vdots \\
\widehat{G_m} &= (\textstyle\sum_{f_i \in G_m} f_i(\boldsymbol{p})) \rhd \widehat{G_{m-1}} \\
&\vdots \\
(\widehat{\{f_k\}} =)\widehat{G_0} &= f_k(\boldsymbol{p})
\end{aligned}
$$

Since all rules in $\mathit{Higher}(f_k)$ are not applicable under σ we deduce that $\sum_{f_i \in G_k} f_i(\boldsymbol{p})$ has no transitions (is deadlocked) for all $1 \leq k \leq m$. Moreover, since $p \rhd q$ behaves like q when p is deadlocked, the above equations imply that $f_k(\boldsymbol{p}) \xrightarrow{a} t$ is one of the transitions of $\widehat{G_m}$, and thus of the auxiliary term, hence:

$$\mathit{AuxiliaryTerm}[f_1(\boldsymbol{p}), \ldots, f_l(\boldsymbol{p})] \xrightarrow{a} t.$$

If part. Assume $\mathit{AuxiliaryTerm}[f_1(\boldsymbol{p}), \ldots, f_l(\boldsymbol{p})] \xrightarrow{a} t$. There is a constant $\widehat{F}$ among those that we have constructed such that $f_k(\boldsymbol{p}) \xrightarrow{a} t$ by rule r_k for either one of the auxiliary $f_k \in F$ where $\widehat{F}$ has equation of type (A.1), or F is just $\{f_k\}$ and has the equation of type (A.3). The second case is proved by just reversing the argument in the "only if" part. In the first case F may contain other auxiliary functions f_i apart from f_k. We deduce that no rule in $\bigcup_{f_i \in F} \mathit{Higher}(f_i)$ is applicable. Hence, no rule in a smaller set $\mathit{Higher}(f_k)$ is applicable. So, we take $\mathit{Higher}(f_k)$ and use the argument from the "only if" part of the proof to construct the set of $\mathit{higher}(r)$, where r is r_k but with f replacing f_k in the source of the rule. Since no rules in $\mathit{higher}(r)$ are applicable we derive $f(\boldsymbol{p}) \xrightarrow{a} t$ by r. $\square$

In Section 5.2 we have seen the derivation of the auxiliary term for several useful

operators. Here we present an artificial operator which requires a non-trivial application of the above procedure to derive the auxiliary term and the auxiliary rewrite rule. Consider operator f defined as follows:

$$\frac{X \xrightarrow{a} X'}{f(X,Y,V,Z) \xrightarrow{a} f(X',Y,V,Z)}\ r_{1a} \qquad \frac{Y \xrightarrow{c} Y'}{f(X,Y,V,Z) \xrightarrow{c} Y'}\ r_{2c}$$

$$\frac{V \xrightarrow{a} V'}{f(X,Y,V,Z) \xrightarrow{a} f(X,Y,V',Z)}\ r_{3a} \qquad \frac{Z \xrightarrow{c} Z'}{f(X,Y,V,Z) \xrightarrow{c} Z'}\ r_{4c}$$

Let R_1, R_2, R_3 and R_4 be the sets of all rules r_{1a}, r_{2c}, r_{3a} and r_{4c}, for all actions a and c in Act, respectively. The ordering on the rules for f is $R_1 \sqsupset R_2$, $R_3 \sqsupset R_4$ and $R_1 \sqsupset R_4$. There are four auxiliary operators arising from the four sets of rules R_i; we name these operators as f_1, f_2, f_3 and f_4, respectively. The initial value of the set S is as follows:

$$\{(\emptyset,\{f_1\}),(\{f_1\},\{f_2\}),(\emptyset,\{f_3\}),(\{f_1,f_3\},\{f_4\})\}$$

The iteration routine in Figure A.1 terminates after two loops and it adds the pair $(\emptyset,\{f_1,f_3\})$ to S resulting in:

$$S = \{(\emptyset,\{f_1\}),\ (\{f_1\},\{f_2\}),\ (\emptyset,\{f_3\}),\ (\{f_1,f_3\},\{f_4\}),\ (\emptyset,\{f_1,f_3\})\}.$$

The set of equations that arise from S is as follows, where we write $\boldsymbol{X}$ for the sequence X,Y,V,Z:

$$\begin{aligned}
AT &= \widehat{\{f_1\}} + \widehat{\{f_3\}} + \widehat{\{f_1,f_3\}} \\
\widehat{\{f_1\}} &= f_1(\boldsymbol{X}) \rhd \widehat{\{f_2\}} \\
\widehat{\{f_2\}} &= f_2(\boldsymbol{X}) \\
\widehat{\{f_3\}} &= f_3(\boldsymbol{X}) \\
\widehat{\{f_1,f_3\}} &= (f_1(\boldsymbol{X}) + f_3(\boldsymbol{X})) \rhd \widehat{\{f_4\}} \\
\widehat{\{f_4\}} &= f_4(\boldsymbol{X})
\end{aligned}$$

Replacing the constants in the equations by the defining terms results in an auxiliary term and the following auxiliary rewrite rule

$$f(\boldsymbol{X}) \rightarrow f_1(\boldsymbol{X}) \rhd f_2(\boldsymbol{X}) \ +\ f_3(\boldsymbol{X}) \ +\ (f_1(\boldsymbol{X}) + f_3(\boldsymbol{X})) \rhd f_4(\boldsymbol{X}).$$

There are several further commonly used process operators that are naturally defined by SOS rules with orderings: action refinement operator [17,40], the two operators that internalise testing [40], the unless operator [10], several "delay" operators from timed process languages [19,29,41] including the operator from Example 2.8, and the timed versions of standard process operators where certain *timed* properties, such as *maximal time synchrony*, hold. The auxiliary terms for all these operators are relatively straightforward.

B Proof of Lemma 5.9

Suppose that f is as in the lemma and i is one of its active arguments. Let $\boldsymbol{p}$ be a vector of n closed terms. We prove both soundness and head normalisation concurrently by induction on the size of terms $f(\boldsymbol{p})$.

We begin with soundness of the rewrite rules introduced by the lemma.

(1) The distributivity rewrite rules $f_{dn(i)}$ are clearly sound. Consider the other distributivity rules $f_{ds(i)}$. Let $\boldsymbol{p}$ be such that p_i is $q+q'$. In order to prove soundness of $f_{ds(i)}$ it suffices to show $f(\boldsymbol{p}) \sim f(\boldsymbol{p})[q/p_i] + f(\boldsymbol{p})[q'/p_i]$ for every rewrite of the form $f(\boldsymbol{p}) \rightarrow f(\boldsymbol{p})[q/p_i] + f(\boldsymbol{p})[q'/p_i]$ which is correct in $\mathcal{P}$. Assume that the following is a correct rewrite:

$$f(\boldsymbol{p}) \rightarrow f(\boldsymbol{p})[q/p_i] + f(\boldsymbol{p})[q'/p_i]$$

Let $f(\boldsymbol{p}) \xrightarrow{a} t$. By (3) for f we deduce $p_i \xrightarrow{a_i} p_i'$ for some p_i'. Hence, $q+q' \xrightarrow{a} p_i'$ and either $q \xrightarrow{a} p_i'$ or $q' \xrightarrow{a} p_i'$. So, we have either $f(\boldsymbol{p})[q/p_i] \xrightarrow{a} t$ or $f(\boldsymbol{p})[q'/p_i] \xrightarrow{a} t$; hence, $f(\boldsymbol{p})[q/p_i] + f(\boldsymbol{p})[q'/p_i] \xrightarrow{a} t$. The other direction follows correspondingly.

(2) Let $\boldsymbol{p}$ be a vector of n closed terms and let $\boldsymbol{a}_I$ be the trigger of a specific rule of type (3) for f. Assume $f(\boldsymbol{a}_I.\boldsymbol{p}) \rightarrow a.C[\boldsymbol{p}]$ is a valid rewrite in $\mathcal{P}$. Then, $f(\boldsymbol{a}_I.\boldsymbol{p}) \xrightarrow{a} C[\boldsymbol{p}]$ by the mentioned rule. Since f is simply distinctive, there is no other rule for f by which we can derive $f(\boldsymbol{a}_I.\boldsymbol{p}) \xrightarrow{a} C[\boldsymbol{p}]$: see Definition 2.10. Hence, $f(\boldsymbol{a}_I.\boldsymbol{p}) \sim a.C[\boldsymbol{p}]$.

(3) Let $\boldsymbol{p}$ be a vector of n closed terms, and $f(\boldsymbol{p}) \rightarrow \mathbf{0}$ be a valid rewrite in $\mathcal{P}$. The ordering $\succ'$ tells us that there is no internal reduction of $f(\boldsymbol{p})$ such that the resulting term can be rewritten by any of the distributivity or action rewrite rules for f. Assume for contradiction that $f(\boldsymbol{p}) \xrightarrow{a} t$ for some a and t. This has to be by one of the rules of type (3). The induction hypothesis gives that all p_i can be rewritten to terms p_i' in head normal form. If one of p_i's is a sum of terms, then one of the distributivity rules $f_{ds(i)}$ can be used to rewrite $f(\boldsymbol{p}')$, contradicting the correctness of $f(\boldsymbol{p}) \rightarrow \mathbf{0}$. If none of the p_i's is a sum, then they are action prefixed terms: $p_i' = a_i.p_i''$ for some p_i''s. Hence, $f(\boldsymbol{p}')$

can be rewritten with the action rewrite rule contradicting the correctness of $f(\boldsymbol{p}) \rightarrow \mathbf{0}$. Hence, $f(\boldsymbol{p}) \stackrel{a}{\not\rightarrow}$ for all actions a, and $f(\boldsymbol{p}) \sim \mathbf{0}$.

Next, we consider head normalisation. We shall prove that there exists a term p in hnf such that $f(\boldsymbol{p}) \twoheadrightarrow p$ in $\mathcal{P}$. By the inductive hypothesis the components of $\boldsymbol{p}$ are in hnf. There are three cases:

(1) One of the terms p_i is $\mathbf{0}$. Then, none of the distributivity and action rewrite rules can be applied to rewrite $f(\boldsymbol{p})$. Hence, $f(\boldsymbol{p}) \rightarrow \mathbf{0}$ by f_{nil}.
(2) One of the terms p_i is $q + q'$ where q and q' are distinct syntactically and not equal to $\mathbf{0}$ closed terms. Then, $f(\boldsymbol{p}) \rightarrow f(\boldsymbol{p})[q/p_i] + f(\boldsymbol{p})[q'/p_i]$ by distributivity. By the induction hypothesis, there exist head normal forms p' and p'' such that $f(\boldsymbol{p})[q/p_i] \twoheadrightarrow p'$ and $f(\boldsymbol{p})[q'/p_i] \twoheadrightarrow p''$, respectively. Clearly, $p' + p''$ is in hnf, or can be rewritten to hnf.
(3) All p_i have the form $b_i.p_i'$. If the actions b_is constitute a trigger for $f(\boldsymbol{p})$, then the appropriate action rewrite rule is used giving a rewrite with the target in hnf. If the actions b_is do not make up a trigger, then the deadlock rewrite rule is used, giving the result.

C Proof of Lemma 5.12

For each operator of G that is not simply distinctive we apply the strategy presented in either Lemma 5.4 or in Lemma 5.10. This gives the required PL G'. The PRS $\mathcal{P}$ is obtained by adding to the PRS for B all the instances of the distributivity, action and deadlock rewrite rules for all simply distinctive operators in G' as required by Lemma 5.9, and all the instances of the auxiliary rewrite rules and the priority resolving rewrite rules as required by Lemma 5.4 and Lemma 5.10. It follows from these lemmas that $\mathcal{P}$ is sound for bisimulation, and it remains to prove that it is head normalising.

We use induction on the structure of terms over G'. In view of the result in Lemma 5.9 it is sufficient to consider only operators f which are not simply distinctive and which are free of implicit copies. Consider $f(\boldsymbol{p})$ with all terms p_i in hnf. There are two cases.

(1) The operator f satisfies the conditions of Lemma 5.10. If p_1 is $\mathbf{0}$, then $f(p_1) \rightarrow \mathbf{0}$ by the deadlock rewrite rule. Otherwise, without loss of generality let p_1 be $\sum_{j\in J} a_j.p_{1j}$. The priority resolving rules allow us to remove summands $a_l.p_{1l}$ if there is a summand $a_k.p_{1k}$ such that $r_{a_k} > r_{a_l}$, where r_{a_k} and r_{a_l} are rules for f with actions a_k and a_l in the premises, respectively. So, $f(p_1)$ is rewritten eventually to $f(\sum_{j\in K} a_j.p_{1j})$ where no further summand can be removed, with at least one remaining. Then, apply the action rewrite rule to obtain hnf.
(2) The operator f satisfies the conditions of Lemma 5.4 but not the conditions of

Lemma 5.10. Hence, $f(\boldsymbol{p}) \rightarrow \mathit{AuxiliaryTerm}[f_1(\boldsymbol{p}), \ldots, f_l(\boldsymbol{p})]$ where operators f_j are the auxiliary simply distinctive operators generated for f by Lemma 5.4. By inspection of the auxiliary term we know that it can be expressed as $t \rhd t'$ where t is a sum form: see Definition 5.3. More specifically, without loss of generality, t is $f_1(\boldsymbol{p}) + \ldots + f_k(\boldsymbol{p})$ where the operators f_i are some of the operators f_j above. Since each f_i is simply distinctive $f_i(\boldsymbol{p})$ can be rewritten to hnf by Lemma 5.9. Hence, term t can be rewritten to hnf. If t can be rewritten to $\mathbf{0}$, then $f(\boldsymbol{p}) \twoheadrightarrow t'$ and t' can be expressed as $t \rhd t'$ for some new t, t' with t that can be rewritten to hnf. If t rewrites to $t_1 + t_2$, then $f(\boldsymbol{p}) \twoheadrightarrow (t_1 + t_2) \rhd t' \rightarrow t_1 \rhd t' + t_2 \rhd t'$ by the distributivity rule for $\rhd$ in Figure 1. If $t \twoheadrightarrow a.u$, for some term u, then $f(\boldsymbol{p}) \twoheadrightarrow (a.u) \rhd t' \rightarrow a.u$ by the action rule for $\rhd$ in Figure 1. As the size of the auxiliary term is finite, $f(\boldsymbol{p})$ rewrites eventually to hnf.

D Proof of Theorem 6.4

We argue that, for a decidable subclass of OSOS PLs, namely syntactically well-founded and linear OSOS PLs, the PRSs generated by algorithm in Figure 2 are strongly normalising, for closed terms modulo associativity and commutativity of $+$ operator.

Proposition D.1 *Let G be a syntactically well-founded and linear OSOS process language. Then, the OSOS process language G' produced for G by the algorithm in Figure 2 is also syntactically well-founded and linear.*

Apart from w and W defined in Section 6, we shall also use other weight functions. In our termination proof we shall use the notion of *marked* terms and operator symbols: $f^{\#}$ is a marked operator if f is an operator. Hence, we extend the definitions of w and W to cover not only $\Sigma_{G'}$ operators but also $\Sigma^{\#}_{G'}$ operators, where $\Sigma^{\#}_{G'} = \{f^{\#} \mid f \in \Sigma_{G'}\}$. Henceforth, the weight functions that we define are over $\Sigma_{G'} \cup \Sigma^{\#}_{G'}$. But first we extend the functions w and W so that they apply to the extended PL G': We set $w(f^c)$ to $w(f)$ for each not free of implicit copies operator $f \in \Sigma$. Also, for each free of implicit copies but not simply distinctive operator f, we set $w(f_i)$ to $w(f)$ for all simply distinctive operators f_is that are created for f in Lemma 5.4.

Definition D.2 Let G be a syntactically well-founded and linear OSOS process language, and let G' be the OSOS process language produced for G by the algorithm in Figure 2. Functions W', e, p, $\mathit{pref} : \mathbb{T}(\Sigma_{G'} \cup \Sigma^{\#}_{G'}) \rightarrow \mathbb{N}$ are as follows:

(1)
- $W'(X) \stackrel{def}{=} 0$,
- $W'(t_1 + t_2) = W'(t_1 +^{\#} t_2) \stackrel{def}{=} max(W'(t_1), W'(t_2))$,
- $W'(t_1 \rhd t_2) = W'(t_1 \rhd^{\#} t_2) \stackrel{def}{=} max(W'(t_1), W'(t_2))$,

- $W'(f(t_1,\ldots,t_n)) \stackrel{def}{=} 1+W'(t_1)+\cdots+W'(t_n)$ if f is any prefixing or marked prefixing operator,
- $W'(f(t_1,\ldots,t_n)) \stackrel{def}{=} w(f)+W'(t_1)+\cdots+W'(t_n)$ otherwise.

(2) For any term t $e(t)$ is 1 if t contains $+$, $+^{\#}$, marked or unmarked non-simply distinctive operators, or marked or unmarked simply distinctive operators that have no active arguments, excluding the deadlocked operator $\mathbf{0}$, and 0 otherwise. Note, that constants and prefixing are among simply distinctive operators that have no active arguments.

(3) Function *pref* is defined by

- $pref(X) \stackrel{def}{=} 0$,
- $pref(f(t_1,\ldots,t_n)) \stackrel{def}{=} p(t_1)+\ldots+p(t_n)$ if f is $+$ or $+^{\#}$,
- $pref(f(t_1,\ldots,t_n)) \stackrel{def}{=} 1+p(t_1)+\ldots+p(t_n)$, if f is a prefixing or marked prefixing operator,
- $pref(f(t_1,\ldots,t_n)) \stackrel{def}{=} w(f)+p(t_1)+\ldots+p(t_n)$ otherwise,

where $p(X) \stackrel{def}{=} 0$, and $p(f(t_1,\ldots,t_n))$ is $1+p(t_1)+\cdots+p(t_n)$ if f is prefixing or marked prefixing operator, and $p(t_1)+\cdots+p(t_n)$ otherwise.

We simply calculate that $W(t) \geq W'(t)$ for all $t \in \mathbb{T}(\Sigma_G)$. Moreover, $W(t) \geq pref(t)$ since $W(t) \geq W'(t)$ and $W'(t) \geq p(t)$.

Definition D.3 Let G be a syntactically well-founded and linear OSOS process language., and let G' be the OSOS process language produced for G by the algorithm in Figure 2. Functions $two^{\#}$, two, $one : \mathbb{T}(\Sigma_{G'} \cup \Sigma^{\#}_{G'}) \to \mathbb{N}$ are as follows:

(1)
- $two^{\#}(X) \stackrel{def}{=} 0$,
- $two^{\#}(f(t_1,\ldots,t_n)) \stackrel{def}{=} two(f(t_1,\ldots,t_n))$ if $f \in \Sigma^{\#}_{G'}$, and 0 otherwise.

(2)
- $two(X) \stackrel{def}{=} 0$,
- $two(f(t_1,\ldots,t_n)) \stackrel{def}{=} 3+\sum_{i=1}^{n} one(t_i)$ if f is a marked non-simply distinctive operator,
- $two(f(t_1,\ldots,t_n)) \stackrel{def}{=} 1+\sum_{i=1}^{n} one(t_i)$ if f is $+$ or $+^{\#}$,
- $two(f(t_1,\ldots,t_n)) \stackrel{def}{=} \sum_{i=1}^{n} one(t_i)$ otherwise.

(3)
- $one(X) \stackrel{def}{=} 0$,
- $one(f(t_1,\ldots,t_n)) \stackrel{def}{=} 1$ if f is $+$ or $+^{\#}$,
- $one(f(t_1,\ldots,t_n)) \stackrel{def}{=} 0$ otherwise.

Definition D.4 An ordering $\sqsupset$ over $\Sigma_{G'} \cup \Sigma^{\#}_{G'}$ terms is defined as follows: $t \sqsupset s$ if and only if

(1) $e(t) > e(s)$, or
(2) $e(t) = e(s)$ and $W'(t) > W'(s)$, or

(3) $e(t) = e(s)$ and $W'(t) = W'(s)$ and $pref(t) > pref(s)$, or
(4) $e(t) = e(s)$ and $W'(t) = W'(s)$ and $pref(t) = pref(s)$ and $two^{\#}(t) > two^{\#}(s)$.

An ordering $\sqsupseteq$ is a union of $\sqsupset$ and $\{(t,s) \mid W'(t) = W'(s)$ and $e(t) = e(s)$ and $pref(t) = pref(s)$ and $two^{\#}(t) = two^{\#}(s)\}$.

We easily check that $\sqsupset$ is transitive and irreflexive, well-founded, and closed under substitution. Clearly, $\sqsupseteq$ is reflexive and transitive and closed under substitution. Moreover, $\sqsupset$ is strictly monotonic and $\sqsupseteq$ is weakly monotonic. Hence, according to the definitions and notation in [23], $\sqsupseteq$ is a *weak reduction order*.

Proof of Theorem 6.4. Assume an OSOS PL $G = (\Sigma, A, R, >)$ which is both linear and syntactically well-founded. Let $G' = (\Sigma', A', R', >')$ be the OSOS PL generated by the algorithm in Figure 2. Moreover, let $\mathcal{P} = (\Sigma', \mathrm{R}, \succ)$ be the PRS produced for G by the algorithm in Figure 2. Since G' is both linear and syntactically well-founded (Proposition D.1) it is sufficient to show that the underlying TRS (Σ', R), denoted by $\mathcal{T}$, is strongly normalising by Proposition 3.4.

We shall employ the *dependency pair* and *dependency graph* techniques due to Arts and Giesl [4]. Since we deal here with rewriting modulo *AC* of $+$ we use the extension of dependency pair and dependency graph techniques to take into account *AC* due to Kusakari and Toyama [23]. Alternatively, we could have employed the *AC* extension due to Marché and Urbain [25]. The basic notions and definitions taken from [23] are as follows. An operator $f \in \Sigma$ is a *defined symbol* if it appears as the head operator of the left-hand side of some rewrite rule in R. An operator $f \in \Sigma$ is a *constructor* if it is not a defined symbol. Next, we define marking of terms [23]: $X^{\#} = X$, $(t_1 + t_2)^{\#} = (t_1)^{\#+} +^{\#} t_2^{\#+}$, and $(f(t_1, \cdots, t_2))^{\#} = f^{\#}(t_1, \cdots, t_2)$ otherwise. Moreover, $X^{\#+} = X$, $(t_1 + t_2)^{\#+} = (t_1)^{\#+} +^{\#} t_2^{\#+}$, and $(g(t_1, \cdots, t_2))^{\#+} = g(t_1, \cdots, t_2)$ otherwise. For example

$$(\mathbf{0} + (f(\mathbf{0} + g) + h))^{\#} \;=\; (\mathbf{0} +^{\#} (f(\mathbf{0} + g) +^{\#} h)$$

The *AC*-dependency pairs are defined as follows. If $f(s_1, \ldots, s_n) \to C[g(t_1, \ldots, t_m)]$ is a rewrite rule of $\mathcal{T}$ with g a defined symbol, then

$$\langle f(s_1, \ldots, s_n)^{\#}, g(t_1, \ldots, t_m)^{\#} \rangle$$

is a *dependency pair* of $\mathcal{T}$. If $s_1 + s_2 \to r$ is a rewrite rule of $\mathcal{T}$, then

$$\langle ((s_1 + s_2) + Z)^{\#}, (r + Z)^{\#} \rangle$$

is an *extended dependency pair* of $\mathcal{T}$, where Z is a fresh variable. Clearly, $\langle ((s_1 + s_2) + Z)^{\#}, (r + Z)^{\#} \rangle = \langle ((s_1 +^{\#} s_2) +^{\#} Z), (r +^{\#} Z) \rangle$, and we shall use this explicit

form from now on. An expression is an *AC-dependency pair* of $\mathcal{T}$ if it is a dependency pair of $\mathcal{T}$ or an extended dependency pair of $\mathcal{T}$.

A sequence of dependency pairs $\langle s_1,t_1\rangle,\langle s_2,t_2\rangle,\ldots$ is an *AC-chain* if there exists a substitution σ such that $\sigma t_j(\stackrel{\#}{\rightarrow})^*\sigma t'_j \trianglerighteq_{hd} \sigma s_{j+1}$ holds for every two consecutive pairs $\langle s_j,t_j\rangle$ and $\langle s_{j+1},t_{j+1}\rangle$ in the sequence. The notions $\stackrel{\#}{\rightarrow}$ and $\trianglerighteq_{hd}$ are defined as follows. Let TRS $R^{\#}$ be $\{(X+Y)+\#\,Z) \rightarrow (X+^{\#}\,Y)+^{\#}\,Z)\}$, and let $t\downarrow_{\#}$ denote the normal form of t in $\rightarrow_{R^{\#}}$ modulo AC of $+$. We define $s \stackrel{\#}{\rightarrow} t$ as $s \rightarrow t'$ and $t = t'\downarrow_{\#}$ for some t'. Informally, the relation $s \trianglerighteq_{hd} t$ means that $s' \in [s]$ and $s' = C[t]$ for some context $C[X]$ such that t appears as an argument of a $+$ term in $C[X]$ and this term is not guarded by any other operator except possibly for $+$. Note, that $p+(q+t)) \trianglerighteq_{hd} t$ but not $f(t+p) \trianglerighteq_{hd} t$. For precise definitions and illustrating examples the reader is referred to [23].

An *AC-dependency graph* of $\mathcal{T}$ is the directed graph whose nodes are the AC-dependency pairs of $\mathcal{T}$ 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 an *AC*-chain.

A weak reduction order $\sqsupseteq$ is a *weak AC-reduction order* if (a) $\sqsupseteq$ is *AC-compatible*, namely if $s \in [t]$, then $s \sqsupseteq t$, and (b) $\sqsupseteq$ has the *AC-deletion property*: $(X+Y)+Z \sqsupseteq X+Y$. Moreover, a quasi ordering $\sqsupseteq$ satisfies the *AC-marked condition* if $(X+Y)+^{\#}\,Z) \sqsupseteq (X+^{\#}\,Y)+^{\#}\,Z)$ and $(X+^{\#}\,Y)+^{\#}\,Z) \sqsupseteq (X+Y)+^{\#}\,Z)$.

We are ready to state the result we will use to prove termination.

Result D.5 [23] A TRS $\mathcal{P}$ is strongly normalising if there exists a weak *AC*-reduction order $\sqsupseteq$ satisfying the *AC*-marked condition such that

(1) $l \sqsupseteq r$ for all rewrite rules $l \rightarrow r$ of $\mathcal{T}$;
(2) $s \sqsupseteq t$ for all dependency pairs $\langle s,t\rangle$ on a cycle of the *AC*-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 *AC*-dependency graph for $\mathcal{T}$.

The required ordering $\sqsupseteq$ will be defined in terms of the weight functions from Definitions D.2 and D.3, which are in turn based on w and W functions from Definition 6.

Remark. Unlike in [14], 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 weight function w is such that $w(f) = 0$ for most of the operators f. Our proof works with $w(f) \geq 0$.

We return to our proof. The only constructors in $\mathcal{T}$ are the prefixing operators $a.$, for all $a \in A$, the operator $\mathbf{0}$ and possibly other constants in Σ (no defining rules).

Next, we work out the *AC*-dependency pairs for $\mathcal{T}$. We begin with the dependency pairs for $\mathcal{B}$. Firstly, there are two extended dependency pairs for our *AC* operator $+$:

$$\langle((X +^{\#} \mathbf{0}) +^{\#} Z), (X +^{\#} Z)\rangle \tag{D.1}$$

$$\langle((X +^{\#} X) +^{\#} Z), (X +^{\#} Z)\rangle \tag{D.2}$$

There are three *AC*-dependency pairs for $\rhd^{\#}$:

$$\langle(X + \mathbf{0}) \rhd^{\#} Z,\ X \rhd^{\#} Z\rangle \tag{D.3}$$

$$\langle(X + Y) \rhd^{\#} Z,\ X \rhd Z\ +^{\#}\ Y \rhd Z\rangle \tag{D.4}$$

$$\langle(X + Y) \rhd^{\#} Z,\ X \rhd^{\#} Z\rangle \tag{D.5}$$

A typical operator $(f, n) \in \Sigma_{G'} \setminus \mathsf{B}$ may have several types of rewrite rules and thus dependency pairs. If f is not free of implicit copies, then $f_{copy} \in \mathrm{R}$ gives rise to

$$\langle f^{\#}(\boldsymbol{X}),\ f^{c\#}(\boldsymbol{Y})\rangle \tag{D.6}$$

where f^c is free of implicit copies operator.

If f is free of implicit copies but not simply distinctive, then there will be a large number of *AC*-dependency pairs arising from the auxiliary rewrite rule from Lemma 5.4. If the auxiliary rewrite rule is $l \to r$, then it produces an *AC*-dependency pair $\langle l^{\#}, r^{\#}\rangle$. Explicitly, this *AC*-dependency pair is

$$\langle f^{\#}(\boldsymbol{X}),\ (\mathit{AuxiliaryTerm}[f_1(\boldsymbol{X}), \cdots, f_l(\boldsymbol{X})])^{\#}\rangle \tag{D.7}$$

There will be *AC*-dependency pairs $\langle l^{\#}, s^{\#}\rangle$, where s a proper subterm of r which is not a variable. Clearly, by Lemma 5.4, the head operator of s is not a constructor. If an *AC*-dependency pair $\langle l^{\#}, s^{\#}\rangle$ occurs in a cycle, it will occur with other *AC*-dependency pairs that we give numbers to. Hence, we will not number then, except for the following type of *AC*-dependency pairs

$$\langle f^{\#}(\boldsymbol{X}),\ f_p^{\#}(\boldsymbol{X})\rangle \tag{D.8}$$

where $p \in \{1, \ldots, l\}$. They may be on cycles involving *AC*-dependency pairs that arise from the action rewrite rules. We shall list them below.

If f is free of implicit copies and simply distinctive, then there will be several types of *AC*-dependency pairs arising from the rewrite rules from Lemma 5.9. The *AC*-dependency pair

$$\langle f^{\#}(\ldots,X_i+Y_i,\ldots),\ f(\ldots,X_i,\ldots)+^{\#} f(\ldots,Y_i,\ldots)\rangle \tag{D.9}$$

can occur in cycles. The *AC*-dependency pairs we are particularly interested in are

$$\langle f^{\#}(X+a.Y+b.Z),\ f^{\#}(X+a.Y)\rangle \tag{D.10}$$

$$\langle f^{\#}(\ldots,X_i+\mathbf{0},\ldots),\ f^{\#}(\ldots,X_i,\ldots)\rangle \tag{D.11}$$

$$\langle f^{\#}(\ldots,X_i+Y_i,\ldots),\ f^{\#}(\ldots,X_i,\ldots)\rangle \tag{D.12}$$

The action rewrite rule gives rise to the *AC*-dependency pair

$$\langle f^{\#}(\boldsymbol{a}_i.\boldsymbol{X}),\ C^{\#}[\boldsymbol{X}]\rangle \tag{D.13}$$

as well as to the following types of *AC*-dependency pairs, where $C[\boldsymbol{X}] \equiv D[g(\boldsymbol{Y})]$ for some context $D[\,]$ and g a defined operator:

$$\langle f^{\#}(\boldsymbol{a}_i.\boldsymbol{X}),\ g^{\#}(\boldsymbol{Y})\rangle \tag{D.14}$$

Notice that there are *AC*-dependency pairs of the types described above for the head operator of $C[\boldsymbol{X}]$ and for g depending on the type the operators. Finally, there is the *AC*-dependency pair arising from the deadlock rewrite $\langle f^{\#}(\boldsymbol{X}),\mathbf{0}^{\#}\rangle$; it clearly cannot occur in any cycle.

With the above listed types of *AC*-dependency pairs we construct the *AC*-dependency graph for $\mathcal{T}$. Here, we shall only identify all possible cycles as it is all that we need by Result D.5.

- There are cycles created by self-embedding *AC*-dependency pairs (D.1)–(D.2), (D.3) and (D.5). If f is simply distinctive and $f \in \Sigma$, and if f occurs in the context $C[\boldsymbol{X}]$ of its action rewrite rule, namely $C[\boldsymbol{X}] \equiv D[f(\boldsymbol{t})]$ by the linearity of f for some context $D[\,]$, then there is a cycle generated by $\langle f^{\#}(\boldsymbol{a}_i.\boldsymbol{X}),\ f^{\#}(\boldsymbol{t})\rangle$: an instance of (D.14).
- There may be cycles that are created by several *AC*-dependency pairs. For example, the shortest cycles are of the form $\langle f^{\#}(\boldsymbol{X}),\ f_p^{\#}(\boldsymbol{X})\rangle$, $\langle f_p^{\#}(\boldsymbol{a}.\boldsymbol{X}), f^{\#}(\boldsymbol{t})\rangle$, where the last pair arises from the action rewrite rule. A bit longer cycles are of the form $\langle f^{\#}(\boldsymbol{X}),\ f^{c\#}(\boldsymbol{Y})\rangle$, $\langle f^{c\#}(\boldsymbol{X}),\ f_p^{c\#}(\boldsymbol{X})\rangle$, $\langle f_p^{c\#}(\boldsymbol{a}.\boldsymbol{X}), f^{\#}(\boldsymbol{t})\rangle$, the last pair arises from the action rewrite rule. There may be longer cycles that involve more than two operators, but they are made up solely from instances of dependency pairs (D.6)–(D.8), (D.10)–(D.12) and (D.14). The common property of all such cycles is that they contain an instance of (D.14).

Now, we need a weak *AC*-reduction order that satisfies the conditions of Result D.5. We argue that $\sqsupseteq$ is the required weak *AC*-reduction order. The order $\sqsupseteq$ is a weak reduction order. It is *AC*-compatible as it equated all *AC*-equivalent terms. It satisfy

the *AC*-deletion property as $two^{\#}((X+Y)+Z) = 2$ and $two^{\#}(X+Y) = 1$. Finally, it satisfies the *AC*-marked condition since e, W', $pref$ and $two^{\#}$ equate the sides of the required pairs.

Next, we show that the ordering $\sqsupseteq$ satisfies the three conditions of Result D.5.

(1) We show that $l \sqsupseteq r$ for every rewrite rule of $\mathcal{T}$. Since function $two^{\#}$ returns 0 for all variables and terms whose head operator is not marked, $two^{\#}(l) = 0 = two^{\#}(r)$ for all our rewrite rules $l \to r$. Hence, we shall not consider $two^{\#}$ further for this case. We begin with rewrite rules for $\mathcal{B}$:
   (a) $+_{dn} : X + \mathbf{0} \sqsupseteq X$ since $e(X+\mathbf{0}) \geq 1 > 0 = e(X)$;
   (b) $+_{ice} : X + X \sqsupseteq X$ since $e(X+X) = 1 > 0 = e(X)$;
   (c) $\rhd_{dis} : (X+Y) \rhd Z \sqsupseteq X \rhd Z + Y \rhd Z$ since $e((X+Y) \rhd Z) = 1 = e(X \rhd Z + Y \rhd Z)$, and $W'((X+Y) \rhd Z) = max(max(0,0),0) = 0$ and $W'(X \rhd Z + Y \rhd Z) = max(max(0,0)max(0,0)) = 0$, and $pref(lhs) = w(\rhd) \geq 0 = pref(rhs)$.
   (d) $\rhd_{act} : a.X \rhd Y \sqsupseteq a.X$ since $e(a.X \rhd Y) = e(a.X)$, and $W'(a.X \rhd Y) = 1 = W'(a.X)$. Also, $pref(lhs) = w(\rhd) + 1 \geq 1 = pref(rhs)$.
   (e) $\rhd_{nil} : X \rhd Y \sqsupseteq Y$ since $e(X \rhd Y) = 0 = e(X)$ and $W'(X \rhd Y) = 0 = W'(Y)$ and $pref(X \rhd Y) = w(\rhd) \geq 0 = pref(Y)$.

   Now we consider rewrite rules for operators $f \in \Sigma' \setminus \Sigma_{\mathsf{B}}$. Let m is the number of active arguments of f. The above comment regarding $two^{\#}$ applies also for the remaining rewrite rules.
   (f) $f_{copy} : f(\boldsymbol{X}) \to f^c(\boldsymbol{Y})$. We have f is simply distinctive if and only if f^c is simply distinctive. We easily check that $f(\boldsymbol{X}) = f^c(\boldsymbol{Y})$.
   (g) $f_{aux} : f(\boldsymbol{X}) \to AuxiliaryTerm[f_1(\boldsymbol{X}), \ldots, f_l(\boldsymbol{X})]$. We have $lhs = rhs$. Since f is not simply distinctive we have $e(lhs) = 1$ and $e(rhs)$ is at most 1. Recall that $w(f) = w(f_i)$ for all $1 \leq i \leq l$. Thus, clearly $W'(lhs) = w(f) = W'(rhs)$. Finally, $pref(lhs) = w(f) = pref(rhs)$.
   (h) $f^a_{pr} : f(X + a.Y + b.Z) \to f(X + a.Y)$. Since $m = 1$ we verify that $e(lhs) = 1 = e(rhs)$ and $W'(lhs) = w(f) + 1 = W'(rhs)$. We have $pref(lhs) = w(f) + 2 > w(f) + 1 = pref(rhs)$.
   (i) $f_{dn}(i) : f(\ldots, X_i + \mathbf{0}, \ldots) \to f(\ldots, X_i, \ldots)$. We have $lhs \sqsupset rhs$ since $e(lhs) = 1$, as it contains $+$, and $e(rhs) = 0$, so $e(lhs) > e(rhs)$.
   (j) $f_{ds}(i) : f(\ldots, X_i + Y_i, \ldots) \to f(\ldots, X_i, \ldots) + f(\ldots, Y_i, \ldots)$. We have $rhs \sqsupseteq lhs$ since $e(lhs) = 1 = e(rhs)$, $W'(lhs) = w(f) = W'(rhs)$ and $pref(lhs) = w(f) \geq 0 = pref(rhs)$.
   (k) $f_{act} : f(\boldsymbol{a}_i.\boldsymbol{X}) \to a.C[\boldsymbol{X}]$. Clearly, $e(lhs) = 1 = e(rhs)$ since either prefixing occurs or f is simply distinctive with no active arguments. We have two cases for f. Assume that f has active arguments, namely $m \geq 1$. We have $W'(f(\boldsymbol{a}_i.\boldsymbol{X})) = w(f) + m \geq W(C[\boldsymbol{X}]) + m$ since $w(f) \geq W(C[\boldsymbol{X}])$ by syntactical well-foundedness. Now, $m + W(C[\boldsymbol{X}]) \geq m + W'(C[\boldsymbol{X}])$ and $\geq W'(a.C[\boldsymbol{X}])$. Moreover, $pref(f(\boldsymbol{a}_i.\boldsymbol{X})) = w(f) + m$ and $w(f) + m \geq W(C[\boldsymbol{X}]) + m \geq W'(C[\boldsymbol{X}]) + m \geq p(C[\boldsymbol{X}]) + m \geq pref(a.C[\boldsymbol{X}])$.

   When f has no active arguments, then $W'(f(\boldsymbol{X})) = w(f) \geq 1 + W(C[\boldsymbol{X}])$

since $w(f) > W(C[\boldsymbol{X}])$. Then, $1+W(C[\boldsymbol{X}]) \geq 1+W'(C[\boldsymbol{X}]) = W'(a.C[\boldsymbol{X}])$. Also, $pref(f(\boldsymbol{X})) = w(f) > W(C[\boldsymbol{X}]) \geq 1 + p(C[\boldsymbol{X}]) = pref(a.C[\boldsymbol{X}])$.

(ℓ) $f_{nil} : f(\boldsymbol{X}) \to \mathbf{0}$. Due to the ordering on transition rules, the rewrite rules is effectively of two forms. Firstly, $f(\boldsymbol{a}.\boldsymbol{X}) \to \mathbf{0}$, where $\boldsymbol{a}.\boldsymbol{X}$ is not a trigger for f and f is not a constant, and secondly $f(\boldsymbol{X}) \to \mathbf{0}$ when f is a constant with no defining rules. Note, that if f is a constant with some defining rules, then the action rewrite rules will always apply and thus the deadlock rewrite rule will never apply.

In both cases above $e(lhs) = 1 > 0 = e(rhs)$, since lhs either involves prefixing or a simply distinctive operator with no active arguments.

(2) We need to show $lhs \sqsupseteq rhs$ for all the AC-dependency pairs $\langle lhs, rhl \rangle$. In part 3 below we prove $lhs \sqsupset rhs$ for the AC-dependency pairs (D.1)–(D.3), (D.5), (D.10)–(D.13). Since (D.13) is more general than (D.14), and (D.7) is more general than other AC-dependency pairs arising from the auxiliary rewrite rule, we shall only consider (D.7). Also, we show $lhs \sqsupseteq rhs$ for (D.4), (D.6) and (D.9).

(D.4): $e(lhs) = 1 = e(rhs)$, $W'(lhs) = 0 = W'(rhs)$, $pref(lhs) = w(\rhd^{\#}) \geq 0 = pref(rhs)$, and $two^{\#}(lhs) = 1 = two^{\#}(rhs)$.

(D.6): Operator f is simply distinctive iff f^c is simply distinctive. Hence, we deduce $two^{\#}(lhs) = two^{\#}(rhs)$ and they are equal to either 3 or 0. The functions e, W' and $pref$ have been calculated in (f) of part 1, so we are done.

(D.7): The functions e, W' and $pref$ have been calculated in (g) of part 1, so we only check $two^{\#}$. Since f is not simply distinctive $two^{\#}(lhs) = 3$. Note, that since all the operators f_i of the rhs are simply distinctive we obtain $two^{\#}(f_i(\boldsymbol{X})) = 0$. If the outermost operator of the rhs is $\rhd^{\#}$, then $two^{\#}(rhs) \leq 2$. Else, namely the outermost operator of the rhs is $+^{\#}$, $two^{\#}(rhs) \leq 3$.

(D.9): The functions e, W' and $pref$ have been calculated in (j) of part 1, so we only check $two^{\#}$. We have $two^{\#}(lhs) = 1 = two^{\#}(rhs)$ as f is simply distinctive.

(3) Each cycle in the AC-dependency graph contains at least one AC-dependency pair of the type (D.1)–(D.3), (D.5), (D.10)–(D.12) and $(D.14)$. Since the longer than 1 cycles contain (D.14), and since (D.13) is more general than (D.14), we show $lhs \sqsupset rhs$ for (D.1)–(D.3), (D.5), (D.10)–(D.13).

(D.1) $\langle((X +^{\#} \mathbf{0}) +^{\#} Z), (X +^{\#} Z)\rangle$: The functions e, W' and $pref$ evaluate to equal values for the lhs and rhs of this AC-dependency pair, but $two^{\#}(lhs) = 2 > 1 = two^{\#}(rhs)$.

(D.2) $\langle((X +^{\#} X) +^{\#} Z), (X +^{\#} Z)\rangle$: As for (D.1).

(D.3) $\langle(X + \mathbf{0}) \rhd^{\#} Z,\ X \rhd^{\#} Z\rangle$: clearly $e(lhs) = 1 > 0 = e(rhs)$, hence $lhs \sqsupset rhs$.

(D.5) $\langle(X + Y) \rhd^{\#} Z,\ X \rhd^{\#} Z\rangle$: As for (D.3).

(D.10) $\langle f^{\#}(X + a.Y + b.Z),\ f^{\#}(X + a.Y)\rangle$: $lhs \sqsupset rhs$. Here, $m = 1$ as there is just one active argument, and although $e(lhs) = e(rhs)$ and $W'(lhs) = W'(rhs)$ we have $pref(lhs) = w(f^{\#}) + 2w(a^{\#}.) > w(f^{\#}) + w(a^{\#}.) = pref(rhs)$ since $w(a^{\#}.) = w(a.) \geq 1$.

(D.11) $\langle f^{\#}(\ldots, X_i + \mathbf{0}, \ldots),\ f^{\#}(\ldots, X_i, \ldots)\rangle$: since $e(lhs) = 1$, as f is simply distinctive and has active arguments, and $e(rhs) = 0$ we have $e(lhs) > e(rhs)$.

(D.12) $\langle f^{\#}(\ldots, X_i + Y_i, \ldots),\ f^{\#}(\ldots, X_i, \ldots)\rangle$: As for (D.11) above.

(D.13) $\langle f^{\#}(\boldsymbol{a}_i.\boldsymbol{X}),\ (C[\boldsymbol{X}])^{\#}\rangle$: We have $e(lhs) \geq e(rhs)$. Moreover, if $m \geq 1$, then we have $W'(f^{\#}(\boldsymbol{a}_i.\boldsymbol{X})) = w(f^{\#}) + m \geq W((C[\boldsymbol{X}])^{\#}) + m \geq W'((C[\boldsymbol{X}])^{\#}) + m$. Clearly, the last term is greater than $W'((C[\boldsymbol{X}])^{\#})$. 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 the syntactical well-foundedness. Since $W((C[\boldsymbol{X}])^{\#}) \geq W'((C[\boldsymbol{X}])^{\#})$ we have $lhs \sqsupset rhs$.

This completes the proof of termination.

References

[1] L. Aceto. Deriving complete inference systems for a class of GSOS languages generating regular behaviours. In B. Jonsson and J. Parrow, editors, *Proceedings of the 5th International Conference on Concurrency Theory CONCUR'94*, volume 836 of *LNCS*, pages 449–464. Springer, 1994. Also, an unpublished full version.

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

[3] L. Aceto, W. 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.

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

[5] F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

[6] 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):127–168, 1986.

[7] J.C.M. Baeten, J.A. Bergstra, J.W. Klop, and W.P. Weijland. Term-rewriting systems with rule priorities. *Theoretical Computer Science*, 67:283–301, 1989.

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

[9] J.C.M. Baeten and C.A. Middelburg. Process algebra with timing: Real time and discrete time. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, pages 627–684. Elsevier Science, 2001.

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

[11] B. Bloom. Structured operational semantics as a specification language. In *Conference Record of the 22nd ACM Symposium on Principles of Programming Languages*, pages 107–117. ACM Press, 1995.

[12] B. Bloom, A. Cheng, and A. Dsouza. Using a protean language to enhance expressiveness in specification. *IEEE Transactions on Software Engineering*, 23:224–234, 1997.

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

[14] D.J.B. Bosscher. Term rewriting properties of SOS axiomatisations. In *Proceedings of International Conference on Theoretical Aspects of Computer Software TACS'94*, volume 789 of *LNCS*, pages 425–439. Springer, 1994.

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

[16] R. Cleaveland and S. Sims. The Concurrency Workbench of New Century. http://www.cs.sunysb.edu/∼cwb/.

[17] R.J. van Glabbeek. *Comparative Concurrency Semantics and Refinement of Actions*. PhD thesis, CWI, 1990.

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

[19] M. Hennessy and T. Regan. A process algebra for timed systems. *Information and Computation*, 117:221–239, 1995.

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

[21] J.R. Kennaway and F.J. de Vries. Infinitary rewriting. In Terese, editor, *Term Rewriting Systems*, pages 668–711. Cambridge University Press, 2003.

[22] 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.

[23] K. Kusakari and Y. Toyama. On proving AC-termination by argument filtering method. *IPSJ Transactions on Programming*, 41(SIG 4 (PRO 7)):65–78, 2000.

[24] L. Léonard and G. Leduc. A formal definition of time in LOTOS. *Formal Aspects of Computing*, 10:248–266, 1998.

[25] C. Marché and X. Urbain. Termination of associative-commutative rewriting by dependency pairs. In T. Nipkow, editor, *Proceedings of the 9th International Conference on Rewriting Techniques and Applications RTA'98*, volume 1379 of *LNCS*, pages 241–255. Springer, 1998.

[26] R. Milner. A complete inference system for a class of regular behaviours. *Journal of Computer System Sciences*, 28:439–466, 1984.

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

[28] R. De Nicola and M. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.

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

[30] D.M. Park. Concurrency on automata and infinite sequences. In P. Deussen, editor, *Conference on Theoretical Computer Science*, volume 104 of *LNCS*, pages 167–183. Springer, 1981.

[31] G. Plotkin. A structural approach to operational semantics. *Journal of Logic and Algebraic Programming*, 60-61:17–141, 2004.

[32] J.C. van de Pol. Operational semantics of rewriting with priorities. *Theoretical Computer Science*, 200:289–312, 1998.

[33] A.W. Roscoe. *The Theory and Practice of Concurrency*. Prentice Hall, 1998.

[34] M. Sakai and Y. Toyama. Semantics and strong sequentiality of priority term rewriting systems. *Theoretical Computer Science*, 208:87–110, 1998.

[35] S. A. Schneider. *Concurrent and Real-time Systems*. Wiley, 2000.

[36] S. Sims. The Process Algebra Compiler. http://www.reactive-systems.com/pac/.

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

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

[39] I. Ulidowski. Priority rewrite systems for OSOS process languages. In R. Amadio and D. Lugiez, editors, *Proceedings of the 14th International Conference on Concurrency Theory CONCUR 2003*, volume 2761 of *LNCS*, pages 87–102. Springer, 2003.

[40] I. Ulidowski and I.C.C. Phillips. Ordered SOS rules and process languages for branching and eager bisimulations. *Information and Computation*, 178(1):180–213, 2002.

[41] I. Ulidowski and S. Yuen. Process languages with discrete time based on the Ordered SOS format and rooted eager bisimulation. *Journal of Logic and Algebraic Programming*, 60-61:401–461, 2004.