Formats of Ordered SOS Rules with Silent Actions

Irek Ulidowski[1] and Iain Phillips[2]

[1] Research Institute for Mathematical Sciences, Kyoto University, Kyoto, Japan
[2] School of Computing, Imperial College, London, England

Abstract. We present a general and uniform method for defining structural operational semantics (SOS) of process algebra operators by traditional Plotkin-style rules equipped with an ordering, the new feature which states the order of application of rules when deriving transitions of process terms. Our method allows to represent negative premises and copying in the presence of silent actions. We identify a number of general formats of unordered and ordered rules with silent actions and show that divergence sensitive branching and weak bisimulation relations are preserved by all operators in the relevant formats. A comparison with the existing formats for branching and weak bisimulations shows that our formats are more general.

1 Introduction

Structured Operational Semantics (SOS) is considered to be the standard method for defining the operational meaning of process operators in an arbitrary process language. It was originated by Milner for CCS [Mil89] and formalised by Plotkin [Plo81]. The meaning of each operator on processes is given by a set of transition rules. Each rule describes how the behaviour of a process (constructed with the operator and some subprocesses) depends on the behaviour of these subprocesses. For example, the rule below is one of the rules for a parallel composition operator. It allows to infer that $a.\mathbf{0} \parallel a.b.\mathbf{0}$ can perform action a since both $a.\mathbf{0}$ and $a.b.\mathbf{0}$ can perform a.

$$\frac{X \xrightarrow{a} X' \quad Y \xrightarrow{a} Y'}{X \parallel Y \xrightarrow{a} X' \parallel Y'}.$$

Process operators can be classified according to the form of rules defining their operational meaning. A format of rules is a collection of forms of rules. We say that an operator is in a certain format if its rules belong to that format, and a process language is in a format if all its operators are in that format.

Most of the popular process operators are in the De Simone format [dS85]. However, De Simone rules do not make use of either the negative behaviour of subprocesses (*negative premises*: the inability to perform actions) or the branching behaviour of processes (*copying*: multiple use of identical process variables). Not surprisingly, there are process operators which cannot be adequately defined by De Simone rules alone. These include, for example, sequential composition, priority, replication and checkpoint operators [Mil89, BW90]. In order to provide for such operators Bloom, Istrail and Meyer proposed the GSOS format [BIM95], which extends the De Simone format with negative premises and copying. This paper provides an alternative method for defining such operators.

An important problem concerning formats of rules is how to use silent actions in rules. Original De Simone and GSOS formats treat both silent and visible actions in the same way, namely as visible. This is unsatisfactory when one wishes to work with *weak equivalences* (where actions may be hidden) since many operators (definable in these formats) do not preserve the considered equivalences. Formats of rules with silent actions were studied by Bloom [Blo90, Blo95], Vaandrager [Vaa91] and the first author [Uli92, Uli94]. A common feature of these approaches is to represent the traditional character of silent actions via τ-rules, proposed in [Blo90]. The motivation is as follows: if f is n-ary operator and the behaviour of $f(\boldsymbol{X})$ depends on the behaviour of its component X_i then when X_i evolves silently $f(\boldsymbol{X})$ can do nothing else but to evolve silently along with X_i. This can be expressed by insisting that the set of rules for f contains for each such X_i a τ-rule of the following form.

$$\frac{X_i \xrightarrow{\tau} X_i'}{f(X_1, \ldots, X_i, \ldots, X_n) \xrightarrow{\tau} f(X_1, \ldots, X_i', \ldots, X_n)}\ \tau_i$$

A notion intimately related to the unobservable character of silent actions is *divergence*. Results in [Uli94, Blo95] show that in a setting with τ-rules if one chooses to equate divergence (infinite sequence of silent actions) and deadlock then rules with negative premises are unacceptable since they can distinguish between the two notions. But, treating divergence as different from deadlock allows one to use rules with negative premises safely [Uli92, Uli94].

In this paper we present a general method for defining process operators by Plotkin-style rules (with no negative premises) which are equipped with an ordering. Our method was informally described in [PU96]. The ordering indicates the order in which rules are applied when deriving transitions of process terms. The behaviour of a process $f(\boldsymbol{p})$ can be determined by examining the rules for f starting with rules highest in the ordering and, if those are not applicable, then considering the lower rules. More generally, our method is similar to the idea of ordering sentences in the field of logic programming to avoid the use of negative information and to ordering rewrite rules in the field of term rewriting. In order to illustrate our method we give an alternative definition of the sequential composition operator ; [BIM95] by the following rule schemas and τ-rules, where a and c are any visible actions.

$$\frac{X \xrightarrow{a} X'}{X;Y \xrightarrow{a} X';Y}\ r_{a*} \qquad\qquad \frac{X \xrightarrow{\tau} X'}{X;Y \xrightarrow{\tau} X';Y}\ \tau_1$$

$$\frac{Y \xrightarrow{\tau} Y'}{X;Y \xrightarrow{\tau} X;Y'}\ \tau_2 \qquad\qquad \frac{Y \xrightarrow{c} Y'}{X;Y \xrightarrow{c} Y'}\ r_{*c}$$

The ordering $>$ on the above rules is such that for all actions a and c we have $r_{a*} > r_{*c}, \tau_2$, and $\tau_1 > r_{*c}, \tau_2$. Hence, $p;q$ can perform an initial action of q (by rule τ_2 or r_{*c}) if neither r_{a*} nor τ_1 are applicable, that is if $p \stackrel{\tau}{\nrightarrow}$ and $p \stackrel{a}{\nrightarrow}$ for all actions a. When p is a totally divergent process, for example defined by a rule $p \stackrel{\tau}{\rightarrow} p$, then q will never start since τ_1 is always applicable.

We argue that any GSOS language can be equivalently formulated in terms of a positive GSOS language equipped with an ordering. This result offers a new approach to developing simple but expressive formats of rules where positive, negative, silent and branching behaviour of processes can be treated consistently.

The contents of our paper are as follows. We start with positive GSOS rules. In order to differentiate between visible and silent actions in rules we insist that silent actions are unobservable and independent of the environment. These two properties are formulated as conditions on positive GSOS rules. We propose another property (and the resulting condition) concerning the use of process resources. Thus, we define two pairs of formats of unordered and ordered positive GSOS rules which satisfy (some of) these conditions. We show that the relevant formats preserve divergence sensitive branching and weak bisimulation preorders. Finally, we argue that our formats are more expressive than the existing formats for these preorders.

The full version of this work [UP96] contains the proofs of our results and more examples illustrating their application.

Acknowledgements. We wish to thank the referees for their comments and suggestions. Thanks are also due to Paul Taylor for his LaTeXmacros.

2 Preliminaries

Let Vis be a finite set of visible actions, ranged over by a, b and c, and $\tau \notin \mathsf{Vis}$ be the silent action. $\mathsf{Vis} \cup \{\tau\}$ is ranged over by α and β. Let Var be a countable set of variables ranged by $X, Y, \ldots$. A signature Σ is a set of operators, namely pairs (f, n) where f is an operator symbol and $n \in \mathbb{N}$ is the arity. When the arity of (f, n) is clear from the context the operator is abbreviated as f. The set of open terms over Σ with variables in $V \subseteq \mathsf{Var}$, denoted by $\mathbb{T}(\Sigma, V)$, is ranged over by $t, t' \ldots$. The set of closed terms, written as $\mathrm{T}(\Sigma)$, is ranged over by $p, q, \ldots$. Σ context with n holes $C[X_1, \ldots, X_n]$, often written as $C[\boldsymbol{X}]$, is a member of $\mathbb{T}(\Sigma, \{X_1, \ldots, X_n\})$. If $t_1, \ldots, t_n$ are Σ terms then $C[t_1, \ldots, t_n]$ is the term obtained by substituting each X_i by t_i. An operator (f, n) preserves a preorder $\sqsubseteq$ if for all vectors of n closed terms $\boldsymbol{t}$ and $\boldsymbol{t}'$ we have $\boldsymbol{t} \sqsubseteq \boldsymbol{t}'$ implies $f(\boldsymbol{t}) \sqsubseteq f(\boldsymbol{t}')$. A *substitution* ρ is a mapping from Var to $\mathrm{T}(\Sigma)$, it extends to a mapping $\mathbb{T}(\Sigma) \rightarrow \mathrm{T}(\Sigma)$ in the standard way. Expressions $t \stackrel{\alpha}{\rightarrow} t'$ and $t \stackrel{\alpha}{\nrightarrow}$, where $t, t' \in \mathbb{T}(\Sigma, V)$, are called *transitions* and *negative transitions* respectively. They are varied over by $T, T', \ldots$ and $nT, nT', \ldots$ respectively.

2.1 GSOS Process Languages

We recall the definitions of the GSOS format, GSOS process languages and other related notions from [BIM95].

Definition 1. A GSOS rule is an expression of the form

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

where all variables X_i and Y_{ij} are distinct, $I, K \subseteq \{1, \ldots, n\}$ and all J_i and L_k are finite subsets of $\mathbb{N}$. $C[\boldsymbol{X}, \boldsymbol{Y}]$ is a context with variables among $\boldsymbol{X}$ and $\boldsymbol{Y}$.

Let r be a GSOS rule for f as in Definition 1. Then, f is the *operator* of r and *rules*(f) is the set of all rules for f. Transitions and negative transitions above the horizontal bar in r are called *premises*, written as *pre*(r). The transition below the bar in r is called the *conclusion*, written as *con*(r). Action α in the conclusion of r is the action of r. $C[\boldsymbol{X}, \boldsymbol{Y}]$ is the target. The set of all α_{ij} in r is denoted by *actions*(r). A rule is *negative* if it contains any $X \stackrel{a}{\not\rightarrow}$ premise, otherwise it is *positive*. The ith argument X_i is *active* in some rule, written as $i \in \mathit{active}(r)$ if it appears in its premises. An argument is active in a set $S \subseteq \mathit{rules}(f)$ if it is active in some rule in S. Overloading the notation denote the set of such i's by *active*(S), and write *active*(f) instead of *active*$(\mathit{rules}(f))$. Consequently, the ith argument of $f(\boldsymbol{X})$ is active if it is active in some rule for f. Rules which are not τ-rules are hereafter called *action* rules.

Definition 2. A GSOS *process language* is a triple $(\Sigma, \mathsf{Act}, R)$, where Σ is a finite set of operators, $\mathsf{Act} \subseteq \mathsf{Vis} \cup \{\tau\}$ and R is a finite set of GSOS rules for operators in Σ.

Given a GSOS process language, a *labelled transition system* can be defined for the language in the standard way as, for example, in [BIM95, GV92, Gro93]. A labelled transition system for $(\Sigma, \mathsf{Act}, R)$ is the structure $(\mathrm{T}(\Sigma), \mathsf{Act}, \rightarrow)$, where $\mathrm{T}(\Sigma)$ is a set of *process terms* or *processes* and $\rightarrow \subseteq \mathrm{T}(\Sigma) \times \mathsf{Act} \times \mathrm{T}(\Sigma)$ is the unique *transition relation* generated by the language.

2.2 Branching and Weak Bisimulation Preorders

We will use some standard abbreviations. We write $p \xrightarrow{\alpha} q$ for $(p, \alpha, q) \in \rightarrow$ and read it as "process p performs α and in doing so becomes q". We write $p \xrightarrow{\alpha}$ when there is q such that $p \xrightarrow{\alpha} q$, and $p \stackrel{\alpha}{\not\rightarrow}$ when for no q we have $p \xrightarrow{\alpha} q$. Expression $p \stackrel{\tau}{\Rightarrow} q$ denotes $p(\xrightarrow{\tau})^* q$ and $p\Uparrow$, read as "p is *divergent*", means $p(\xrightarrow{\tau})^\omega$. We say p is convergent, written as $p\Downarrow$, if p is not divergent. Finally, if $\alpha = \tau$ then $p \xrightarrow{\hat{\alpha}} p'$ means $p \xrightarrow{\tau} p'$ or $p \equiv p'$, else it is simply $p \xrightarrow{\alpha} p'$.

Definition 3. Assume a labelled transition system $(\mathrm{T}(\Sigma), \mathsf{Act}, \rightarrow)$. A binary relation R over $\mathrm{T}(\Sigma)$ is a *branching bisimulation* if pRq implies

(a) $\forall \alpha.\ p \xrightarrow{\alpha} p'$ implies $(\exists q', q''.\ q \stackrel{\tau}{\Rightarrow} q' \xrightarrow{\hat{\alpha}} q'' \wedge pRq' \wedge p'Rq'')$

(b) $p\Downarrow$ implies $q\Downarrow$ and
$\forall \alpha.\ q \xrightarrow{\alpha} q'$ implies $(\exists p', p''.\ p \stackrel{\tau}{\Rightarrow} p' \xrightarrow{\hat{\alpha}} p'' \wedge p'Rq \wedge p''Rq')$

$p \underset{\sim}{\sqsubset}_{BB} q$ if there exists a branching bisimulation R such that pRq.

A binary relation R over $\mathrm{T}(\Sigma)$ is a *weak bisimulation* if R is defined as branching bisimulation but without conditions pRq' and $p'Rq$. $p \underset{\sim}{\sqsubset}_{WB} q$ if there exists a weak bisimulation R such that pRq.

Example 1. Consider CCS-like processes $p \equiv b.\mathbf{0}+\tau.a.\mathbf{0}$, $q \equiv p+a.\mathbf{0}$ and $r \equiv p+\Omega$, where Ω is defined by $\Omega \xrightarrow{\tau} \Omega$. By Definition 3, we have $\underset{\sim}{\sqsubset}_{BB} \subseteq \underset{\sim}{\sqsubset}_{WB}$. Moreover, $p \underset{\sim}{\sqsubset}_{WB} q$, but $p \not\underset{\sim}{\sqsubset}_{BB} q$ as $q \xrightarrow{a}$, $p \stackrel{\tau}{\Rightarrow} p' \xrightarrow{a}$ and clearly $q \not\underset{\sim}{\sqsubset}_{BB} p'$. Also, $p \not\underset{\sim}{\sqsubset}_{WB} r$ but $r \underset{\sim}{\sqsubset}_{WB} p$.

It is clear that $\underset{\sim}{\sqsubset}_{BB}$ and $\underset{\sim}{\sqsubset}_{WB}$ are preorders. Our branching bisimulation is a possible generalisation of the standard notion as, for example, in [vG90, BW90]. We make the relation sensitive to divergence in the same way as was done with weak bisimulation in [Mil81, Abr87]. Preorder $\underset{\sim}{\sqsubset}_{WB}$ is a version of weak bisimulation relation studied in [Mil81, Abr87, Wal90, Uli94], where testing, modal logic and axiomatic characterisations were proposed and a congruence result with respect to the ISOS format was proved. For processes with no divergence $\underset{\sim}{\sqsubset}_{WB}$ coincides with *delay* bisimulation [BW90, vG90]. We have chosen this finer version of weak bisimulation in preference to the standard [Mil89] because there are process operators, like the action refinement in Section 5, which do not preserve the standard version (the problem is not due to the initial silent actions).

3 Ordered Positive GSOS Rules

The premises of GSOS rules may contain both positive and negative transitions. We propose ordered positive GSOS rules as an alternative, and possibly more concise, method for expressing full GSOS rules. Our method was informally introduced in the workshop paper [PU96]. Here, we repeat the definition of an ordering on rules and state expressiveness results, which did not appear in the original reference.

Definition 4. Let $<_f$ be a transitive relation on $rules(f)$. $r <_f r'$ is interpreted as r having a lower priority than r' (and r' having a higher priority than r) when deriving the transitions of terms with f as the outermost operator. The ordering $<_f$ specifies that a rule can only be applied when no rules with higher priority can be applied. Given a positive GSOS language with a signature Σ, the ordering $<_\Sigma$, or simply $<$, is defined as $\bigcup_{f \in \Sigma} <_f$. An ordered process language is a tuple $(\Sigma, \mathsf{Act}, R, <)$, where $(\Sigma, \mathsf{Act}, R)$ is a positive GSOS process language and $<$ is the ordering on its rules.

In the next subsection we will argue that for each ordered process language there is an equivalent (full) GSOS language and vice versa, where two process languages are equivalent if they give rise to isomorphic transition systems. Thus, a transition system for an ordered process language is the transition system for the equivalent GSOS language. The transition relation associated with an ordered process language can also be defined directly [PU96, UP96].

3.1 Expressiveness

We show that (full) GSOS languages can be alternatively formulated as equivalent ordered positive GSOS languages and vice versa.

Firstly, we describe a translation of a GSOS language $G = (\Sigma, A, R)$ to an ordered positive GSOS language $H = (\Sigma', A', R', <)$. We set $\Sigma' = \Sigma$ and $A' = A$. Let $(f, n) \in \Sigma$ be defined by the set of rules R_f. Also, let $R_f = R_f^+ \cup R_f^-$, where R_f^+ and R_f^- are sets of positive and negative rules for f respectively. Assume that r is one of the negative rules for f with the form as in Definition 1. Then, r is translated into the set of positive GSOS rules $R_f^{+'}(r)$ which consists of the rule r' and the rules $r'_{\beta_{kl}}$, one for each β_{kl} in r, defined below.

$$\frac{\{X_i \xrightarrow{\alpha_{ij}} X_{ij}\}_{i \in I, j \in J_i}}{f(\boldsymbol{X}) \xrightarrow{\alpha} u}\ r' \qquad\qquad \frac{X_k \xrightarrow{\beta_{kl}} X_{kl}}{f(\boldsymbol{X}) \xrightarrow{\beta_{kl}} t}\ r'_{\beta_{kl}}$$

The term t above is an arbitrary fixed term which does not appear in the target of the conclusion of any rule in R_f, otherwise it might be a valid rule in R_f^+. Note that if any other rule in R has a premise $X_j \stackrel{\beta_{kl}}{\nrightarrow}$ then the set of corresponding rules for f in H will contain the rule $r'_{\beta_{kl}}$. The ordering on rules satisfies $r'_{\beta_{kl}} > r'$, for all appropriate β_{kl}. This guarantees that r' is applicable if $X_i \xrightarrow{\alpha_{ij}} X_{ij}$ and $X_k \stackrel{\beta_{kl}}{\nrightarrow}$, for all suitable α_{ij} and β_{kl}. Moreover, we require that $r'_{\beta_{kl}} > r'_{\beta_{kl}}$, for all β_{kl}. The last condition rules out the possibility of ever using any of $r'_{\beta_{kl}}$ to derive new transitions. Hence, the set of rules for f in H, written as R'_f, is defined as

$$R'_f = \bigcup \{R_f^{+'}(r) \mid r \in R_f^-\} \cup R_f^+$$

It is easily checked that G and H generate the same transition system.

If G has any negative rules then clearly its ordered version H has more rules. However, as far as the amount of computation required (measured in the total number of transitions and negative transitions which need to be checked) in order to derive a transition it is easy to see that it is the same in G and H.

A translation from ordered positive GSOS languages to GSOS languages is also straightforward. As before the sets of operators and actions of H and G are the same. This time we denote the set of rules of H as R and the corresponding set in G as R'. Let $r \in R$ be one of the rules for f. We show how to define the set $R'_f(r)$ of ordinary GSOS rules for f which correspond to r. If $higher(r) = \emptyset$ then $R'_f(r) = \{r\}$, where $higher(r) = \{r'' \mid r'' > r\}$. Otherwise, assume $higher(r) = \{r_i \mid 1 \leq i \leq m\}$ and for each i $pre(r_i) = \{T_{ij} \mid 0 \leq j \leq m_i\}$. According to Definition 4 r can be applied if $pre(r)$ is valid and each $pre(r_i)$ is not valid. Assume that none of r_i is an axiom rule. Thus,

$$R'_f(r) = \{r' \mid con(r') = con(r) \wedge pre(r') = pre(r) \cup \{nT_{ij} \mid \forall i\, \exists j.\, nT_{ij} = \neg T_{ij}\}\}$$

where nT_{ij}'s denote negative premises and $\neg(X \xrightarrow{a} Y)$ means $X \stackrel{a}{\nrightarrow}$. As before, the languages G and H produce isomorphic transition systems.

We easily calculate that $R'_f(r)$ has $\prod_{i=1}^{m} m_i$ rules. Thus, when m and some m_i are greater than 1 it is clear that the fragment of the definition of f consisting of r and $higher(r)$ is more concise than the corresponding fragment $R'_f(r)$.

4 Silent Actions and Formats of Rules

In this section we show how silent actions can be safely introduced in formats of ordered rules. We propose several conditions on the structure of rules and on the orderings which guarantee that silent actions keep their traditional meaning. We identify several formats of rules and prove that weak and branching bisimulation preorders are preserved by the operators definable in the relevant formats.

Notation. In order to shorten the presentation of the forthcoming conditions we leave out the outermost universal quantifiers binding $f \in \Sigma$ and $r, r' \in rules(f)$, where appropriate.

4.1 Branching and Weak Bisimulation Formats

We think that τ-rules embody the independent of the environment character of τ actions [Uli92]: "if the ith argument X_i can contribute to the behaviour of $f(\boldsymbol{X})$ then the silent behaviour of X_i becomes the silent behaviour of $f(\boldsymbol{X})$". In our framework, only active arguments are contributing arguments. This principle can be expressed as

$$\text{if } i \in active(f) \;\text{ then }\; \tau_i \in rules(f) \tag{1}$$

Operators which do not satisfy (1), for example the CCS choice and the left-merge of ACP, are not well behaved: they do not preserve weak bisimulation.

Insisting that all operators have their required τ-rules does not represent the full character of silent actions yet. We additionally require that silent actions are unobservable which, after [Uli92][3], can be interpreted as "silent, unobservable behaviour of the components of a process cannot produce a visible behaviour of the process or a change of its structure". This principle can be formulated as

$$\text{if } \tau \in actions(r) \;\text{ then }\; r \text{ is a } \tau\text{-rule} \tag{2}$$

meaning that no rules except τ-rules can have τ actions in the premises.

We claim that all operators defined by positive GSOS rules which satisfy the above two conditions preserve branching bisimulation. In other words, all operators which can be defined by positive GSOS action rules with no τ's in the premises together with the required τ-rules preserve branching bisimulation.

However, the described class of operators is strictly larger than the class of positive ISOS operators [Uli92, Uli94], thus there is no guarantee that its members preserve weak bisimulation. The operators which do not preserve weak

[3] Action rules with $X \xrightarrow{\tau} X'$ in the premises are allowed in [Vaa91] but must be accompanied by exactly the same rules except with $X \xrightarrow{*} X'$ instead of $X \xrightarrow{\tau} X'$.

bisimulation are those which make the full use of copying. Consider operators *a-and-b*, *a-then-b* and *then-b* from [Uli92]:

$$\frac{X \xrightarrow{a} X' \quad X \xrightarrow{b} X''}{a\text{-}and\text{-}b(X) \xrightarrow{c} \mathbf{0}} \qquad \frac{X \xrightarrow{a} X'}{a\text{-}then\text{-}b(X) \xrightarrow{\tau} then\text{-}b(X)} \qquad \frac{X \xrightarrow{b} X'}{then\text{-}b(X) \xrightarrow{c} \mathbf{0}}$$

The first two rules have multiple occurrences of X in the premises together with the target, in other words they have copies of X. Consider a positive GSOS rule r as in Definition 1, i.e. with $K = \emptyset$. Multiple occurrences of process variables in r can be divided into *explicit* and *implicit* copies. Explicit copies are the multiple occurrences of Y_{ij}'s and X_i's, for $i \notin I$, in the target of r. Implicit copies are the multiple occurrences of X_i's in the premises and (not necessarily multiple) occurrences of X_i's, when $i \in I$, in the target of r. We notice that the first and second of the above rules have implicit copies of X. Consider processes p and q in Example 1. One can easily check that $a\text{-}and\text{-}b(p) \stackrel{c}{\not\Rightarrow}$ but $a\text{-}and\text{-}b(q) \stackrel{c}{\Rightarrow}$. Also, $a\text{-}then\text{-}b(p) \stackrel{c}{\not\Rightarrow}$ but $a\text{-}then\text{-}b(q) \stackrel{c}{\Rightarrow}$. Thus, operators defined by rules with implicit copies can distinguish between weak bisimilar processes but not, we claim, between branching bisimilar processes.

The above discussion concerns the use of process resources in general and the use of process variables in rules in particular. Results in [Uli94] show that operators with linear use of process resources preserve weak bisimulation. In the setting of SOS rules, an argument X_i is used linearly in a rule r if and only if whenever it appears in the premises of r then (a) it appears there at most once and (b) it does not appear in the target of the conclusion of r. In other words, linear use of process variables means no implicit copies. This suggests the third condition. Given a positive GSOS rule r, let *implicit-copies*(r) stand for the set of all variables which have implicit copies in r. The condition is as follows:

$$implicit\text{-}copies(r) = \emptyset \tag{3}$$

Definition 5. A set of positive GSOS rules is called bb (branching bisimulation) if its rules satisfy conditions (1) and (2). A set of positive GSOS rules is called wb (weak bisimulation) if its rules satisfy conditions (1), (2) and (3). An operator is bb (wb) if it is defined by bb (wb) rules. A format of rules is bb (wb) if it consists of bb (wb) rules. A process language is bb (wb) if it only contains bb (wb) process operators.

Note that bb rules allow both explicit and implicit copies but wb rules allow only explicit copies. In order to compare the wb and bb formats (and the formats in the next subsection) with the ISOS format we recall the definition of the ISOS format. A set of rules is in the ISOS format if it consists of ISOS rules defined below, their associated τ-rules and no other rules. An ISOS rule has the form

$$\frac{\{\ X_i \xrightarrow{a_i} X_i'\ \}_{i \in I} \qquad \{\ X_k \stackrel{\tau}{\not\rightarrow} \stackrel{b_{kl}}{\not\rightarrow}\ \}_{k \in K, l \in L_k}}{f(X_1, \ldots, X_n) \xrightarrow{\alpha} C[\boldsymbol{Y}],}$$

where all X_i and X_i' are different variables, $I, K \subseteq \{1, \ldots, n\}$ and all L_k are finite subsets of natural numbers. $C[\boldsymbol{Y}]$ contains at most the variables $Y_1, \ldots, Y_n$, where $Y_i = X_i'$ if $i \in I$ and $Y_i = X_i$ otherwise. Negative transitions are called *refusal* transitions. It is easy to check that the wb format coincides with the positive ISOS format and the bb format is an extension of the positive ISOS format with implicit copying.

Finally, we are ready to state the main result of this subsection.

Theorem 6. All bb (wb) operators preserve branching (weak) bisimulation.

4.2 Branching and Weak Bisimulation Ordered Formats

In this subsection we consider ordered bb and wb rules. Careless orderings on such rules can change the unobservable and independent of the environment character of silent actions. For example, when an action rule r for f is above its ith τ-rule then, for a given $f(\boldsymbol{p})$ such that r is applicable, it may happen that $f(\boldsymbol{p})$ may not be able to perform τ even though p_i can do τ. We present two examples illustrating this problem and derive two conditions which guarantee the traditional character of silent actions.

Consider a parallel composition operator $\|$ defined by the following rule schemas together with τ-rules τ_1 and τ_2 which are not presented.

$$\frac{X \xrightarrow{a} X'}{X \parallel Y \xrightarrow{a} X' \parallel Y}\ r_{a*} \qquad\qquad \frac{Y \xrightarrow{a} Y'}{X \parallel Y \xrightarrow{a} X \parallel Y'}\ r_{*a}$$

If the ordering is $r_{a*} > \tau_2$, for all r_{a*}, then trace equivalent $a.b.\mathbf{0}$ and $a.\tau.b.\mathbf{0}$ can be distinguished by $\|$. For $c.\mathbf{0} \parallel a.b.\mathbf{0} \stackrel{ab}{\Rightarrow}$ but $c.\mathbf{0} \parallel a.\tau.b.\mathbf{0} \stackrel{ab}{\not\Rightarrow}$ since after a action c has a preference over τ. Thus, the first condition might be: if r is a τ-rule then $higher(r) = \emptyset$. The intuition is that τ-rules should not have lower priority. But, although the condition is natural it is also quite restrictive. Consider a binary operator f such that the behaviour of $f(p, q)$ initially depends on the behaviour of the first subprocess (like in the case of sequential composition). This may result in some rules associated with the first argument being above τ_2. We can allow such orderings provided that all the rules, which are above τ_2, are also above all the rules with active second argument. The resulting condition is

$$\text{if } i \in active(f) \cap active(r') \text{ and } r > \tau_i \text{ then } r > r' \tag{4}$$

However, there are operators definable by bb rules satisfying condition (4) which are not well behaved. Consider the priority operator θ (cf [BW90]) which gives d priority over b. It is defined by the rule schema r_α below, for $\alpha \in \{b, d\}$, with the τ-rule τ_1 and the ordering $r_d > r_b$.

$$\frac{X \xrightarrow{\alpha} X'}{\theta(X) \xrightarrow{\alpha} \theta(X')}\ r_\alpha$$

Let $p = b.\mathbf{0} \parallel \tau.d.\mathbf{0}$ and $q = b.\mathbf{0} \parallel d.\mathbf{0}$, where $\parallel$ is the usual interleaved parallel operator. Clearly p and q are trace equivalent but we have $\theta(p) \stackrel{b}{\Rightarrow}$ and $\theta(q) \stackrel{b}{\not\Rightarrow}$. To repair this problem it is enough to require $\tau_1 > r_b$. Thus, we arrive at

$$\text{if } r > r' \text{ and } i \in \mathit{active}(r) \text{ then } \tau_i > r' \tag{5}$$

The intuition here is that in order to apply the rule r we need to make sure that no other rule with a higher priority (and thus their τ-rules) can be applied.

Conditions (4) and (5) are sufficient to ensure that operators which are defined by bb (or wb) rules with an ordering satisfying these conditions preserve branching (weak) bisimulation. Before we state this result we propose a small generalisation of the ordered wb rules. We remind that condition (3) forbids implicit copies in wb rules. However, when wb rules are used with an ordering the condition can be considerably relaxed. We propose to allow implicit copies in wb rules provided that they are below their associated τ-rules:

$$\text{if } \; i \in \mathit{implicit\text{-}copies}(r) \; \text{ then } \; \tau_i > r \tag{6}$$

Consider the $a\text{-}and\text{-}b$ operator defined in the previous subsection with its rules (one action rule and two τ-rules) satisfying the last condition. Then process $a\text{-}and\text{-}b(p)$, for any p, can perform c if the τ-rule for $a\text{-}and\text{-}b$ cannot be applied, in other words if $p \stackrel{\tau}{\not\rightarrow}$, $p \stackrel{a}{\rightarrow}$ and $p \stackrel{b}{\rightarrow}$. Hence, for processes p and q in Example 1 we have $a\text{-}and\text{-}b(p) \stackrel{c}{\not\Rightarrow}$, $a\text{-}and\text{-}b(q) \stackrel{c}{\not\Rightarrow}$ and $a\text{-}and\text{-}b(p) \sqsubseteq_{WB} a\text{-}and\text{-}b(q)$ as expected.

Definition 7. A set of bb rules with an ordering is called bbo (branching bisimulation ordered) if the ordering satisfies conditions (4) and (5). A set of wb rules with an ordering is called wbo (weak bisimulation ordered) if the ordering satisfies conditions (4)-(6). bbo (or wbo) operators, formats of rules and process languages are defined as the corresponding notions in Definition 5.

Theorem 8. All bbo (wbo) operators preserve branching (weak) bisimulation.

5 Applications

An alternative definition of the *sequential composition* operator appears in the Introduction. *Priorities* and *broadcast parallel* operators as well as the *copy+refusal testing system* are defined by ordered SOS rules in [PU96, UP96].

Action refinement is an operation which replaces all occurrences of an action by some process. It is known that for sequential processes action refinement preserves branching bisimulation but not the standard weak bisimulation [vG90]. Below, we define a wbo version of action refinement operator ref_a such that $\mathit{ref}_a(p, q)$ refines all a in p by q. The rules and rule schemas for ref_a are given below, where $b \in \mathsf{Vis} \setminus \{a\}$ and the required τ-rules are not shown.

$$\frac{X \stackrel{b}{\rightarrow} X'}{\mathit{ref}_a(X,Y) \stackrel{b}{\rightarrow} \mathit{ref}_a(X',Y)}\; r_b \qquad\qquad \frac{X \stackrel{a}{\rightarrow} X'}{\mathit{ref}_a(X,Y) \stackrel{\tau}{\rightarrow} \mathit{aux}(X',Y,Y)}\; r_a$$

$$\frac{Y \xrightarrow{\alpha} Y'}{aux(X,Y,Z) \xrightarrow{\alpha} aux(X,Y',Z)}\ q_{*\alpha} \qquad\qquad \frac{}{aux(X,Y,Z) \xrightarrow{\tau} ref_a(X,Z)}\ q_{\alpha *}$$

The ordering satisfies $q_{\alpha *} < q_{*\alpha}$ together with the conditions for wbo rules.

The last example concerns process languages with discrete time. The *maximal progress* property [Wan91, HR95] can be expressed as $p \xrightarrow{\tau}$ implies $p \stackrel{\sigma}{\not\rightarrow}$, where $X \xrightarrow{\sigma} X'$ denotes the passage of one time unit. It means that the process will block the passage of time when it is not stable. Consider a discrete time process language L which satisfies maximal progress and extend it with the CCS parallel $|$. Let $r_{a\bar{a}}$ denote the synchronisation rule for $|$. Then, the rule below specifies the passage of time for $|$ and the ordering $r_\sigma < r_{a\bar{a}}$ guarantees that the maximal progress property holds for the extended language L.

$$\frac{X \xrightarrow{\sigma} X' \quad Y \xrightarrow{\sigma} Y'}{X|\,Y \xrightarrow{\sigma} X'|\,Y'}\ r_\sigma$$

6 Comparison With Related Formats

Firstly, we compare formats for our version of weak bisimulation and the standard weak bisimulation.

- The wb format coincides with the positive ISOS format. Also, the simply WB cool format [Blo95] for the standard weak bisimulation is like the wb format except that it also requires other τ-rules apart from those requested by condition (1).
- The wbo format extends the wb format with stable implicit copying and refusal transitions in the premises of action and τ-rules.

Although the wbo and ISOS formats do not allow arbitrary implicit copying, it is argued in [Uli92] that the branching behaviour captured by rules with implicit copying can also be captured by ISOS rules, and thus by wbo rules. The idea is that instead of using implicit copies of process resources we produce their copies first (by applying rules with explicit copying) and only then we use them. The fully WB cool format [Blo95] allows rules with implicit copying but only when several kinds of auxiliary rules are present. The effect of these auxiliary rules amounts to what we have informally described above: firstly making copies of process resources and then using them independently.

Finally, we consider formats for branching bisimulation.

- The bb format extends the positive ISOS format with implicit copying. It is very similar to fully BB cool format [Blo95].
- The bbo format extends the bb format with refusal transitions in the premises of rules. It extends the fully BB cool format with negative premises.

Thus, our formats of ordered rules for weak and branching bisimulation preorders are more general than the previously proposed formats for these preorders.

References

[Abr87] S. Abramsky. Observation equivalence as a testing equivalence. *Theoretical Computer Science*, 53:225–241, 1987.

[BIM95] B. Bloom, S. Istrail, and A.R. Meyer. Bisimulation can't be traced. *Journal of ACM*, 42(1), 1995. Also appeared as Technical Report TR 90-1150, Cornell, 1990.

[Blo90] B. Bloom. Strong process equivalence in the presence of hidden moves. Preliminary report, MIT, 1990.

[Blo95] B. Bloom. Structural operational semantics for weak bisimulations. *Theoretical Computer Science*, 146:27–68, 1995.

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

[dS85] R. de Simone. Higher-level synchronising devices in MEIJE-SCCS. *Theoretical Computer Science*, 37:245–267, 1985.

[Gro93] J.F. Groote. Transition system specifications with negative premises. *Theoretical Computer Science*, 118, 1993.

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

[HR95] M. Hennessy and T. Regan. A process algebra for timed systems. *Information and Computation*, 117, 1995.

[Mil81] R. Milner. A modal characterisation of observable machine behaviours. In G. Astesiano and C. Böhm, editors, *CAAP 81*, pages 25–34, Berlin, 1981. Springer-Verlag. LNCS 112.

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

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

[PU96] I.C.C. Phillips and I. Ulidowski. Ordered SOS rules and weak bisimulation. In A. Adalat, S. Jourdan, and G. McCusker, editors, *Theory and Formal Methods 1996*, London, 1996. Imperial College Press.

[Uli92] I. Ulidowski. Equivalences on observable processes. In *Proceedings of the 7th Annual IEEE Symposium on Logic in Computer Science*, Santa Cruz, California, 1992.

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

[UP96] I. Ulidowski and I.C.C. Phillips. Formats of ordered SOS rules with silent actions. Technical report, RIMS, Kyoto University, 1996. Available at http://www.kurims.kyoto-u.ac.jp/~irek/.

[Vaa91] F.W. Vaandrager. On the relationship between process algebra and input/output automata. In *Proceedings of the 6th Annual IEEE Symposium on Logic in Computer Science*, Amsterdam, 1991.

[vG90] R.J. van Glabbeek. *Comparative Concurrency Semantics and Refinement of Actions.* PhD thesis, CWI, 1990.

[Wal90] D. Walker. Bisimulation and divergence. *Information and Computation*, 85(2), 1990.

[Wan91] Y. Wang. *A Calculus of Real Time Systems.* PhD thesis, Chalmers University of Technology, Göteborg, Sweden, 1991.