Ordered SOS Process Languages for Branching and Eager Bisimulations

Irek Ulidowski [a], Iain Phillips [b,1]

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

[b] Department of Computing, Imperial College, 180 Queen's Gate, London SW7 2BZ, United Kingdom

Abstract

We present a general and uniform method for defining structural operational semantics (SOS) of process operators by traditional Plotkin-style transition rules equipped with *orderings*. This new feature allows one to control the order of application of rules when deriving transitions of process terms. Our method is powerful enough to deal with rules with negative premises and copying. We show that rules with orderings, called *Ordered SOS* rules, have the same expressive power as GSOS rules. We identify several classes of process languages with operators defined by rules with and without orderings in the setting with silent actions and divergence. We prove that branching bisimulation and eager bisimulation relations are preserved by all operators in process languages in the relevant classes.

Key words: Structured Operational Semantics (SOS), Process languages, Formats of SOS rules: Ordered SOS format, GSOS format, Process preorders: branching bisimulation, eager bisimulation, Congruence results

1 Introduction

Structural Operational Semantics (SOS) is considered to be the standard method for defining the operational meaning of process operators in an ar-

Email addresses: `iu3@mcs.le.ac.uk` (Irek Ulidowski), `iccp@doc.ic.ac.uk` (Iain Phillips).

URLs: `http://www.mcs.le.ac.uk/~iulidowski` (Irek Ulidowski), `http://www.doc.ic.ac.uk/~iccp` (Iain Phillips).

[1] Funded by EPSRC grant GR/K54663

bitrary process language. It was originated by Milner for CCS [27] and formalised by Plotkin [35]. The meaning of each process operator is given by a set of *transition rules*, which have the form $\frac{\text{premises}}{\text{conclusion}}$. The conclusion of a transition rule describes the behaviour of a process constructed with the operator and several component processes. This behaviour depends on the behaviour of the component processes, which is given in the premises of the rule. For example, the following rule is one of the rules for a version of parallel composition operator:

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

This rule allows us to infer that the process $a.\mathbf{0} \parallel a.b.\mathbf{0}$ can perform action a and evolve to the process $\mathbf{0} \parallel b.\mathbf{0}$ since both subprocesses $a.\mathbf{0}$ and $a.b.\mathbf{0}$ can perform a.

Process operators can be classified according to the form of rules defining their operational meaning. A *format* of rules is, informally, a set of forms of rules. We say that an operator is in a certain format if its rules belong to that format. Moreover, a process language is in a certain format if all its operators are in that format. An n-ary operator is said to *preserve* a process relation, for example *strong bisimulation* [30,27], if it produces two related processes from every two sets of n subprocesses that are pairwise related. A process language preserves a relation if all its operators preserve the relation, and a format preserves a relation if all operators which can be defined in the format preserve the relation.

Most of the popular process operators are in the De Simone format [38]. However, De Simone rules, for example the rule above, have rather a restricted form. They do not make use of either the negative behaviour of subprocesses, namely the inability to perform actions, or the branching behaviour of processes. Not surprisingly, there are process operators which cannot be defined by De Simone rules alone. The most important examples of these are sequential composition and priority operators, and replication and checkpoint operators [27]. Such operators are usually defined by rules with *negative premises* (expressions like $X \stackrel{a}{\not\to}$) and *copying* (multiple use of identical process variables) respectively. In order to provide for such operators Bloom, Istrail and Meyer proposed in [14,15] a general format of rules, called the GSOS format, which extends the De Simone format with negative premises and copying. Since then a rich theory has been developed for GSOS languages which includes congruence results for strong bisimulation and ready simulation relations [14,11,15], procedures for automatic generation of complete axiom systems [3,2] and denotational models [5,39].

An important problem concerning formats of transition rules is how to use silent actions in rules. The 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 visible actions may be hidden and thus become silent actions) since many operators which are definable in these formats do not preserve weak equivalences. Formats of rules with positive premises and silent actions were studied by Bloom [12,13], Vaandrager [46] and the first author [40–42], who also considered rules with negative premises. A common feature of these approaches is to represent the *uncontrollable* and *independent of the environment* character of silent actions via special rules called τ-rules due to Bloom [12]. The motivation for τ-rules is the following: If the behaviour of $f(\boldsymbol{X})$, where f is an n-ary process operator, depends on the behaviour of its component X_i, then when X_i performs silent actions $f(\boldsymbol{X})$ can do nothing else but perform silent actions along with X_i. This principle can be expressed in the setting of formats of rules by insisting that the set of rules for f contains, for each such ith component, a τ-rule τ_i of the following form:

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

A notion intimately related to the uncontrollable character of silent actions is *divergence*. In this paper we identify divergence with the ability to perform an infinite sequence of silent actions. Results in [40,41,13] show that, in a setting with τ-rules, if one chooses to equate divergence and deadlock, then rules with negative premises are unacceptable since they can distinguish between the two notions. On the other hand, treating divergence as different from deadlock allows one to use rules with negative premises safely, but under one condition [40,41]: Since the inability to perform an action is not observable in the presence of divergence we do not allow premises of the form $X \stackrel{a}{\nrightarrow}$; but the inability to perform an action in a stable state is observable [26,31], so premises of the form $X \stackrel{\tau}{\nrightarrow}\stackrel{a}{\nrightarrow}$ are allowed. Since we are seeking formats of rules which combine silent actions, negative premises and copying we shall distinguish between divergence and deadlock. As a consequence, we shall work with divergence sensitive versions of eager bisimulation [26,1,48,41,43–45], and branching bisimulation [19,20] relations as the chosen weak equivalences.

It is not obvious how to combine silent actions with negative premises and copying in transition rules so that the resulting format preserves the chosen weak equivalence. There seem to be two different approaches to solve this problem. One can choose the equivalence first and then find a general format which preserves it. The discovered format may be somewhat complicated since it is mainly influenced by a (technical) requirement that all operators definable in the format preserve the chosen equivalence. Alternatively, one may express certain properties of concurrent systems as conditions on the form of transition rules for the operators. For example, the independent of the environment character of silent actions is expressed above as a condition demanding the presence of τ-rules among the defining rules for process operators. Such conditions can be used as a recipe to define formats of rules. Then, one checks which weak equivalences are preserved by these formats. It may happen that simple and intuitive formats which are found by the second approach do not preserve some of the favourite equivalences, but only their minor variations. Ultimately, it is up to the user to choose between possibly complicated formats for standard equivalences and simple formats for somewhat non-standard, but possibly more suitable, equivalences.

The first of the approaches described above was followed in [12,46,13]. For example, one of the equivalences considered by Bloom in [13] is the rooted version of weak bisimulation (or observational congruence) [27,9], but the discovered format proved quite complicated. On the other hand, formats for branching bisimulation equivalences [13] have quite natural formulations. The second approach was followed by the first author who defined the ISOS format [40,41] which is the most general format with silent actions, negative premises and copying so far. ISOS operators do not preserve the standard weak bisimulation [27] but preserve a slightly finer, divergence sensitive version, called *eager bisimulation*, discussed in [26,1,48,41,43]. In this paper we adopt the second approach and develop several general formats of rules for eager and branching bisimulation preorders.

1.1 Results

We present a general method for defining process operators by Plotkin-style transition rules (with no negative premises) which are equipped with orderings. Our method was informally described in [34] and further developed in [43] and also in [45]. An ordering on rules 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 which are highest in the ordering and then considering lower rules. Thus, a rule lower in the ordering can only be applied if none of the rules above it can. Intuitively, this has the effect of applying rules with negative premises. More generally, our method is similar to the idea in the field of logic programming of ordering sentences in order to avoid using negative information. Also, it is similar to ordering rewrite rules in the field of term rewriting systems [8].

We illustrate our method by giving an alternative definition of the sequential composition operator ';' discussed in [14,15]. The rule schemas for ';' are given below, where α and β are any visible or silent actions, and $r_{\alpha\star}$ and $r_{\star\alpha}$ are the names of rule schemas.

$$\frac{X \stackrel{\alpha}{\to} X'}{X;Y \stackrel{\alpha}{\to} X';Y}\ r_{\alpha\star} \qquad\qquad \frac{Y \stackrel{\alpha}{\to} Y' \quad \{X \stackrel{\beta}{\nrightarrow} \mid \text{all } \beta\}}{X;Y \stackrel{\alpha}{\to} Y'}\ r_{\star\alpha}$$

Our version of ';' is defined by the set of rules that contains the following rule schemas

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

where a and c are any visible actions, and an ordering relation $<$ that satisfies, among others, $r_{*c} < r_{a*}$. The set of rules and the ordering must also satisfy several simple and intuitive conditions. These conditions relate to the use of silent actions in the defining rules, and how this influences the ordering. In the case of ';' the conditions require that the set of rules also contains two τ-rules

$$\frac{X \xrightarrow{\tau} X'}{X;Y \xrightarrow{\tau} X';Y}\ \tau_1 \qquad\qquad \frac{Y \xrightarrow{\tau} Y'}{X;Y \xrightarrow{\tau} X;Y'}\ \tau_2$$

and no other rules, and that the ordering additionally satisfies $\tau_2 < r_{a*}$, $r_{*c} < \tau_1$ and $\tau_2 < \tau_1$. As a result we have that $p;q$ can perform an initial action of q (by rule τ_2 or r_{*c}) if none of the rules r_{a*} and τ_1 is applicable, that is if $p \stackrel{\tau}{\nrightarrow}$ and $p \stackrel{c}{\nrightarrow}$ for all c. When p is a totally divergent process, for example defined by rule $p \xrightarrow{\tau} p$, then q will never start since τ_1 is always applicable.

We show that any GSOS process operator can be equivalently defined by positive GSOS rules equipped with an ordering. Therefore, we have a new method, which is an alternative to the GSOS method, for defining simple and expressive process languages where silent and visible, positive and negative, and linear and branching aspects of process behaviour can be represented consistently. Our method is intended to guide the definitions of process operators, particularly those operators which are sensitive to the silent and negative behaviour of their operands.

Furthermore, we define several formats of rules with and without orderings, and thus several classes of process languages, and prove that eager and branching bisimulation preorders are preserved by the relevant formats (and process languages). A comparison with the previously proposed formats, and classes of process languages, for similar preorders shows that our formats and classes are more general.

1.2 Outline

In Section 2 we recall the definition of GSOS rules. We also present a general method for defining process operators by Ordered SOS rules (or OSOS rules, for short), which are positive GSOS rules equipped with orderings. In Section 3 we define the notions of formats and process languages. We discuss GSOS process languages and OSOS process languages, and we argue that GSOS and OSOS languages have the same expressive power. We also give several other expressiveness results concerning subclasses of GSOS and OSOS languages.

We differentiate between visible and silent actions in rules by insisting that silent actions are unobservable and independent of the environment. In Section 4 these two properties are formulated as conditions on the structure of unordered and ordered positive GSOS rules. We also introduce two more properties which concern the treatment of process resources and, for these properties, we propose conditions on copying in rules. Thus, we define two pairs of formats of unordered and ordered positive GSOS rules which satisfy some of these conditions. These formats give rise to four classes of process languages. We show that eager and branching bisimulation preorders are preserved in the relevant classes. Section 5 contains a comparison of the existing formats and classes of process languages. We argue that our formats are more expressive than the existing formats for similar preorders. Section 6 demonstrates the usefulness of ordered rules in defining process operators which usually require rules with negative premises.

2 Rules and Transition Relations

In this section we recall the definition of GSOS rules and GSOS operators, and introduce positive GSOS rules equipped with orderings and operators defined by such rules. We also define transition relations associated with process languages which contain the mentioned types of operators.

2.1 Preliminaries

Var is a countable set of variables ranged over by $X, X_i, Y, Y_i, \ldots$. Σ_n is a set of operators with arity n. A signature Σ is a collection of all Σ_n and it is ranged over by $f, g, \ldots$. The members of Σ_0 are called *constants*; $\mathbf{0} \in \Sigma_0$ is the deadlocked process 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 with n holes $C[X_1, \ldots, X_n]$ is a member of $\mathbb{T}(\Sigma, \{X_1, \ldots, X_n\})$. If $t_1, \ldots, t_n$ are Σ terms, then $C[t_1, \ldots, t_n]$ is the term obtained by substituting t_i for X_i for $1 \leq i \leq n$.

We will use bold italic font to abbreviate the notation for sequences. 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

$\boldsymbol{pRq}$ 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. A preorder $\sqsubseteq$ on $\mathrm{T}(\Sigma)$ is a precongruence if $\boldsymbol{p} \sqsubseteq \boldsymbol{q}$ implies $C[\boldsymbol{p}] \sqsubseteq C[\boldsymbol{q}]$ for all $\boldsymbol{p}$ and $\boldsymbol{q}$ of length n and all Σ contexts $C[\boldsymbol{X}]$ with n holes. Similarly, an operator $f \in \Sigma_n$ preserves $\sqsubseteq$ if, for $\boldsymbol{p}$ and $\boldsymbol{q}$ as above, $\boldsymbol{p} \sqsubseteq \boldsymbol{q}$ implies $f(\boldsymbol{p}) \sqsubseteq f(\boldsymbol{q})$. Consequently, a format preserves $\sqsubseteq$ if all operators definable in the format preserve $\sqsubseteq$. A process language preserves $\sqsubseteq$ if all its operators preserve $\sqsubseteq$. A class of process languages preserves $\sqsubseteq$ if all languages in the class preserve $\sqsubseteq$.

A *substitution* is a mapping $\mathsf{Var} \to \mathrm{T}(\Sigma)$. Substitutions are ranged over by ρ and ρ' and 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 $1 \leq i \leq n$.

$\mathsf{Act} = \mathsf{Vis} \cup \{\tau\}$, ranged over by $\alpha, \beta, \ldots$, is a finite set of actions, where Vis is the set of visible actions, ranged over by $a, b, \ldots$, and τ is a silent, internal action not in Vis.

2.2 GSOS Rules

In this and the next subsection we present two methods, one of them new, for assigning the operational meaning to process operators of an arbitrary process language. Given a process language with the signature Σ and the set of actions A, we describe how to generate a *transition relation* which represents the behaviour of process terms in the language.

GSOS rules [14,15] and τ-rules [12,46,40] are defined as follows:

Definition 1 A GSOS rule is an expression of the form

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

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

Definition 2 A GSOS rule of the form as below is called the τ-rule for f and its ith argument X_i. It is denoted as τ_i^f, or simply as τ_i if f is clear from the context.

$$\frac{X_i \stackrel{\tau}{\to} X_i'}{f(X_1, \ldots, X_i, \ldots, X_n) \stackrel{\tau}{\to} f(X_1, \ldots, X_i', \ldots, X_n)}$$

Expressions $t \xrightarrow{\alpha} t'$ and $t \stackrel{\beta}{\nrightarrow}$, where $t, t' \in \mathbb{T}(\Sigma, V)$, are called *transitions* and *negative transitions* respectively. Transitions are ranged over by $T, T', \ldots$. If transition T stands for $X \xrightarrow{\alpha} X'$, we will sometime write $\neg T$ to denote $X \stackrel{\alpha}{\nrightarrow}$. A (negative) transition which involves only closed terms is called a closed (negative) transition. GSOS rules are ranged over by $r, r', r_\alpha \ldots$.

Next, we define several notations related to GSOS rules.

Definition 3 Let r be a GSOS rule for an operator f as in the above definition. Then, f is the *operator* of r and the elements of $\boldsymbol{X}$ are the arguments of r. We write *rules*(f) for the set of GSOS rules for f. Transitions and negative transitions above the horizontal bar in r are called *premises*. The set of premises is written as *pre*(r). The transition below the bar in r is the *conclusion* and is written as *con*(r). Action γ in the conclusion of r is the *action* of r. $C[\boldsymbol{X}, \boldsymbol{Y}]$ is the *target* of r, denoted by *target*(r). The set of all α_{ij} in r is denoted by *actions*(r). A rule is *negative* if it contains any $X \stackrel{a}{\nrightarrow}$ premise; otherwise it is *positive*. The operation of removing all negative premises from a GSOS rule is denoted by $(\ )^\dagger$. A rule r is an *action* rule if $\tau \notin$ *actions*(r). The ith argument X_i is *active* in rule r, written as $i \in$ *active*(r), if it appears in its premises. An argument is active in a set $S \subseteq$ *rules*(f) if it is active in some rule in S. Overloading the notation, we denote the set of all such i by *active*(S), and write *active*(f) instead of *active*$($*rules*$(f))$. Consequently, the ith argument of $f(\boldsymbol{X})$ is active if it is active in some rule for f.

Next, we define the notions of *implicit* and *explicit copies* of process variables in GSOS rules. This will be needed in establishing the general structure of rules for operators that preserve eager and branching bisimulations.

Informally, copies are the multiple occurrences of the process variables in rules. Consider the following rule r_h:

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

This rule is a recipe for working out all the a-derivatives of process terms of the form $h(p_1, p_2, p_3, p_4)$ (this will be formalised in Section 2.4). Intuitively, in order to derive the a transition of $h(p_1, p_2, p_3, p_4)$, specified by the conclusion of r_h, the arguments p_1, p_2, p_3 and p_4 must have transitions described in the premises. For example, p_1 (as substituted for X_1) must be able to perform both a_{11} and a_{12}. In order to verify this we need two copies of p_1. Since we start with only one copy of each argument in $h(p_1, p_2, p_3, p_4)$ the rule implicitly assumes that there are further copies of the first argument. Moreover, the rule implicitly assumes that there are two copies of the second argument: one of them is used to verify that p_2 can perform a_{21} and another one is needed as the first argument of g. Therefore, copies of variables like X_1 and X_2 will be called implicit.

There are also multiple occurrences of variables X_3, X_4 and Y_{11} in r_h. We notice that X_4 is not used in verifying the premises and that it is simply passed to g. Therefore, only one instance of it is needed and so we do not count multiple occurrences of variables like X_4 as copies. As for X_3 and Y_{11}, we explicitly make copies of these variables as we apply the rule. Copies of variables like X_3 and Y_{11} will be called explicit. The transition a is the cost of making these copies, whereas copies of X_1 and X_2 are there for 'free'. We will show later that some operators with rules with implicit copies do not preserve branching and eager bisimulations, but operators with rules with explicit copies preserve these relations.

Definition 4 Let r be a GSOS rule for an operator f as in the Definition 1. We say that r has *implicit* copies of variable Z the ith argument X_i if (a) Z is an active argument in r, and (b) it appears in the premises of r more than once or it appears in the target of r. We say that r has *explicit* copies of variable Z if (a) Z is a non-active argument in r or $Z = Y_{ij}$ for some Y_{ij}, and (b) Z appears more than once in the target of r.

We notice, by Definition 4, that r_h has implicit copies of X_1 and X_2, and explicit copies of X_3 and Y_{11}. The following notation will be useful later on:

$$\textit{implicit-copies}(r) = \{i \mid \text{there are implicit copies of } X_i \text{ in } r\}$$

Hence, $\textit{implicit-copies}(r_h) = \{1, 2\}$. Also, given a GSOS rule, we will need to differentiate between the sets of implicit copies of process variables in the premises and in the target. Overloading the notation, we define these sets as

$$\textit{implicit-copies}(\textit{pre}(r)) = \{i \mid i \in \textit{implicit-copies}(r) \wedge X_i \notin \textit{Var}(\textit{target}(r))\}$$

$$\textit{implicit-copies}(\textit{target}(r)) = \textit{implicit-copies}(r) \setminus \textit{implicit-copies}(\textit{pre}(r))$$

Clearly, $\textit{implicit-copies}(\textit{pre}(r_h)) = \{1\}$ and $\textit{implicit-copies}(\textit{target}(r_h)) = \{2\}$.

Definition 5 A set of rules is GSOS if it consists of GSOS rules only. An operator f is GSOS if $\textit{rules}(f)$ is GSOS.

All operators of CCS [27], CSP [25] and ACP [10,9] are, or can be defined to be, GSOS.

2.3 Ordered SOS Rules

The premises of GSOS rules may contain both positive and negative transitions. In this section we propose Ordered SOS rules [34,43] as an alternative and equally expressive method for defining process operators. We motivate our idea by a simple example.

Example 6 Consider an operator f defined by the following rules.

$$\frac{X \xrightarrow{a} Y}{f(X) \xrightarrow{a} t} \qquad \frac{X \xrightarrow{c} Y \quad X \stackrel{a}{\not\rightarrow}}{f(X) \xrightarrow{c} t'}$$

It is clear by the first rule that $f(p)$ can perform a if p can perform a. Furthermore, by the second rule, $f(p)$ can perform c if its component p can perform c and cannot perform a. To say that p cannot perform a is equivalent in this case to saying that the first rule cannot be used to derive transitions of $f(p)$. Thus, $f(p) \xrightarrow{c}$ if $p \xrightarrow{c}$ and the first rule cannot be used to derive transitions of $f(p)$. This suggests that instead of the above rules operator f can be defined by positive rules

$$\frac{X \xrightarrow{a} Y}{f(X) \xrightarrow{a} t} \qquad \frac{X \xrightarrow{c} Y}{f(X) \xrightarrow{c} t'}$$

with a condition that the second rule is only applied when the first rule cannot be applied. In other words, we introduce a binary relation on positive rules which defines the *order* of their application when deriving transitions of terms with f as the outermost operator.

The ordering specifies the order in which the rules are to be applied when deriving transitions of terms. The following definition makes this precise:

Definition 7 Let R be a set of GSOS rules. An *ordering* on R is any binary relation on R. A set of rules with an ordering is called Ordered SOS (or OSOS, for short) if it contains positive GSOS rules only. Given an operator f, let $<_f$ denote an ordering on *rules*(f). The operator f is OSOS if *rules*(f) with $<_f$ is OSOS. Let Σ be a set of OSOS operators. The ordering on rules for the operators in Σ, written as $<$, is given by $\bigcup_{f \in \Sigma} <_f$.

We will write *higher*(r) for $\{r' \mid r <_f r'\}$.

2.4 Transition Relations and Labelled Transition Systems

Definition 8 A *labelled transition system* (or LTS, for short) is a structure $(\mathcal{P}A, \rightarrow)$, where $\mathcal{P}$ is the set of processes, A is the set of actions and $\rightarrow \subseteq \mathcal{P} \times A \times \mathcal{P}$ is a *transition relation*.

Given a set Σ of GSOS operators, which come with a set of GSOS rules, and a set A of actions (large enough to contain all the actions mentioned in the rules for Σ), the GSOS rules determine a unique transition relation

$\rightarrow \subseteq \mathrm{T}(\Sigma) \times A \times \mathrm{T}(\Sigma)$ (see, for example, [14,15,21]), and thus we obtain an LTS $(\mathrm{T}(\Sigma), A, \rightarrow)$.

Next, we describe how to associate a transition relation with a set of OSOS operators[2]. Our method builds upon the ideas in [21]. Let $d : \mathrm{T}(\Sigma) \rightarrow \mathbb{N}$ be a function which specifies the depth of ground terms over Σ. Function d is defined inductively as follows: $d(p) = 0$ if p is a constant, and $d(f(p_1, \ldots, p_n)) = 1 + \max\{d(p_i) \mid 1 \leq i \leq n\}$ otherwise.

Definition 9 Let Σ be a set of OSOS process operators with the set of rules R, and let A be a set of actions which contains all the actions which are mentioned in the rules in R. We associate to Σ and A a transition relation, $\rightarrow \subseteq \mathrm{T}(\Sigma) \times A \times \mathrm{T}(\Sigma)$, which is defined as $\rightarrow = \bigcup_{l<\omega} \rightarrow^l$, where transition relations $\rightarrow^l \subseteq \mathrm{T}(\Sigma) \times A \times \mathrm{T}(\Sigma)$ are defined as follows:

$$\begin{aligned} p \xrightarrow{\alpha} p' \in \rightarrow^l \quad \text{if} \quad d(p) = l \quad \text{and} \quad & \exists r \in R, \rho.(\rho(con(r)) = p \xrightarrow{\alpha} p' \\ \text{and} \quad & \rho(pre(r)) \subset \textstyle\bigcup_{k<l} \rightarrow^k \\ \text{and} \quad & \forall r' \in higher(r).\ \rho(pre(r')) \not\subset \textstyle\bigcup_{k<l} \rightarrow^k) \end{aligned}$$

The definition states that a rule r can be used to derive a transition $p \xrightarrow{\alpha} p'$ if $p \xrightarrow{\alpha} p'$ coincides with the conclusion of r under a substitution ρ, all premises of r are valid under ρ and no rule in $higher(r)$ is applicable. The last means that each rule in $higher(r)$ has a premise which is *not* valid under ρ:

Definition 10 Let $r \in rules(f)$ and $pre(r) = \{X_i \xrightarrow{\alpha_{ij}} Y_{ij} \mid i \in I, j \in J_i\}$. Rule r *applies* to $f(\boldsymbol{u})$ if and only if $u_i \xrightarrow{\alpha_{ij}}$ for all relevant i and j. Rule r is *enabled* at term $f(\boldsymbol{u})$ if and only if r applies to $f(\boldsymbol{u})$ and all rules in $higher(r)$ do not apply.

Definition 11 Let $(\mathcal{P}, A, \rightarrow)$ be a LTS, and let $p \in \mathcal{P}$. The set of *immediate derivatives* of p, written as $Der_{\rightarrow}(p)$ or simply as $Der(p)$ if $\rightarrow$ is clear from the context, is defined as $\{(a, p') \mid p \xrightarrow{a} p'\}$. Any process $p' \in Der_{\rightarrow}(p)$ is called an a-derivative of p.

We say that a transition $t \xrightarrow{\alpha} t'$ is *valid* under a substitution ρ in a transition relation $\rightarrow'$ if $(\rho(t), \alpha, \rho(t')) \in \rightarrow'$. Similarly, given processes p and q and action α, $p \xrightarrow{\alpha} q$ is valid in $\rightarrow'$ if $(p, \alpha, q) \in \rightarrow'$. We shall write $p \xrightarrow{\alpha} q$ for $(p, \alpha, q) \in \rightarrow$ when the considered transition system with its transition relation $\rightarrow$ are clear from the context.

[2] Alternatively, we could use the transition relation associated with a set of GSOS operators to which a given set of OSOS operators can be equivalently translated. The translation is defined in the next section.

Finally, in order to make the paper self-contained, we recall the definition of *strong bisimulation* [30,27].

Definition 12 Assume an LTS $(\mathcal{P}, A, \rightarrow)$. A relation $R \subseteq \mathcal{P} \times \mathcal{P}$ is a *strong bisimulation* if whenever pRq, then the following properties hold:

$$\forall p', \alpha.[p \xrightarrow{\alpha} p' \ \text{ implies } \ \exists q'.(q \xrightarrow{\alpha} q' \ \text{ and } \ p'Rq')]$$

$$\forall q', \alpha.[q \xrightarrow{\alpha} q' \ \text{ implies } \ \exists p'.(p \xrightarrow{\alpha} p' \ \text{ and } \ p'Rq')]$$

Strong bisimulation equivalence $\sim$: $p \sim q$ if there exists a strong bisimulation R such that pRq.

It is easy to show that strong bisimulation equivalence is an equivalence.

3 Languages and Expressiveness

In the last section we have discussed how to define process operators by GSOS and OSOS rules, and how to obtain transition relations from the rules. In this section we compare GSOS and OSOS in terms of expressive power. A general framework for comparing the expressive power of different process languages was formulated by Shapiro [36,37]. Our translations can be fitted into this framework, in that they are language homomorphisms which preserve an equivalence, namely strong bisimulation equivalence (Definition 12). However, we opt for an informal presentation.

Process operators can be classified according to how their operational meaning is defined. In this paper, we employ transition rules with orderings and we also use conditions on the structure of rules and on the type of orderings; more traditionally, only transition rules are used, for example De Simone rules or GSOS rules. We shall loosely call this transition rule-based method for giving operational semantics a format.

Definition 13 A *format* is a transition rule-based method for assigning operational meaning to process operators.

The first format was proposed by De Simone [38]. De Simone rules are GSOS rules without negative premises and copying. The GSOS [14,15] format is due to Bloom, Istrail and Meyer [14,15]. The first author developed the ISOS format [40,41], a subformat of GSOS, which treats silent actions τ as unobservable and permits explicit copying and *refusals* (expressions of the form $X \stackrel{\tau}{\not\rightarrow}\stackrel{a}{\not\rightarrow}$) in the premises of rules. There are several more general formats of rules, for example the *tyft/tyxt* and *ntyft/ntyxt* formats [22,21] and the *panth* format [47].

Definition 14 The *OSOS format* consists of positive GSOS rules equipped with orderings on rules.

Process languages whose operators are definable by rules in the GSOS format are very expressive; most of the operators in existing process languages are GSOS operators. This is due to the generality of GSOS rules, which allow both negative premises and copying. In fact, the OSOS format has the same expressive power as the GSOS format.

Definition 15 A *process language* is a triple $(\Sigma, A, \rightarrow)$, where Σ is a signature, A is a set of actions, and $\rightarrow \subseteq \mathrm{T}(\Sigma) \times A \times \mathrm{T}(\Sigma)$ is a transition relation (given by a set of rules with or without orderings).

Often, instead of process language, we shall simply say language.

Definition 16 Given a format $\mathcal{F}$ and a language $\mathcal{P} = (\Sigma, A, \rightarrow)$, we say that $\mathcal{P}$ is in the $\mathcal{F}$ format, or equivalently $\mathcal{P}$ is an $\mathcal{F}$ language, if all operators in Σ are definable within $\mathcal{F}$. A *class* of languages, or simply a class, is a set of languages. The $\mathcal{F}$ class is the set of all $\mathcal{F}$ languages.

We show that GSOS languages can be alternatively formulated as OSOS languages, and conversely. We shall prove this by giving OSOS definition to GSOS operators and by giving GSOS definitions to OSOS operators. However, we begin by introducing a couple of auxiliary notions and motivating examples.

So far we have given alternative definitions in terms of OSOS rules for the sequential composition operator ';' and the operator f from Example 6: both definitions are intuitive and natural. In general, we can give intuitive OSOS definitions to *natural* GSOS operators. We define this notion below.

Throughout this work, we make a simplifying assumption that arguments of process operators are named consistently in the rules for the operators. By this we mean that, given an n-ary operator f, if the kth argument of f in a rule is X, for $k \leq n$, then in all other rules for f the kth argument is also X. Recall that the operation of removing all negative premises from GSOS rules is denoted by $(\ )^{\dagger}$.

Definition 17 Let g be a GSOS operator and let $R = \mathit{rules}(g)$. A relation $\prec : R \times \mathsf{Var} \times (\mathsf{Vis} \cup \{\tau\}) \times R$ is defined by $r \overset{X\,\alpha}{\prec} r'$ if and only if $X \overset{\alpha}{\nrightarrow} \in \mathit{pre}(r)$ and $\mathit{pre}(r'^{\dagger}) = \{X \overset{\alpha}{\rightarrow} X'\}$ for some X'.

The above definition says that two rules r and r' for a GSOS operator are related by $\prec$ via an argument X and an action α if r has the negative premise $X \overset{\alpha}{\nrightarrow}$ and the only positive premise of r' is $X \overset{\alpha}{\rightarrow} X'$ for some X'.

Definition 18 Let g be a GSOS operator and let $R = \mathit{rules}(g)$. We say that

g is natural if

- none of the rules in R has contradictory premises, namely for every rule $r \in R$ if $X \stackrel{\alpha}{\not\rightarrow} \in pre(r)$, then $X \stackrel{\alpha}{\rightarrow} X' \notin pre(r)$ for all X', and
- for each rule $r \in R$ with $X \stackrel{\alpha}{\not\rightarrow} \in pre(r)$ there is $r' \in R$ such that $r \stackrel{X\,\alpha}{\prec} r'$, in other words $pre(r'^{\dagger}) = \{X \stackrel{\alpha}{\rightarrow} X'\}$ for some X'.

A GSOS language is natural if all its operators are natural.

We easily check that the sequential composition operator from the Introduction, operator f from Example 6 and the priority operator from Section 6 are all natural. Another example of a natural GSOS operator is given below.

Example 19 Let h be defined by the following rules:

$$\frac{X \stackrel{a}{\rightarrow} Y \quad X \stackrel{b}{\not\rightarrow}}{h(X) \stackrel{a}{\rightarrow} t}\ r_a \qquad\qquad \frac{X \stackrel{b}{\rightarrow} Y \quad X \stackrel{a}{\not\rightarrow}}{h(X) \stackrel{b}{\rightarrow} t'}\ r_b$$

Neither of the above rules has contradictory premises. We have $r_a \stackrel{X\,a}{\prec} r_b$ and $r_b \stackrel{X\,b}{\prec} r_a$. Thus, h is natural. It can be redefined by the OSOS rules

$$\frac{X \stackrel{a}{\rightarrow} Y}{h(X) \stackrel{a}{\rightarrow} t}\ r'_a \qquad\qquad \frac{X \stackrel{b}{\rightarrow} Y}{h(X) \stackrel{b}{\rightarrow} t'}\ r'_b$$

with the ordering defined by $r'_a < r'_b$ and $r'_b < r'_a$. Notice that $<$ is irreflexive, but it is *not* transitive.

Not all GSOS operators are natural. The following example illustrates this and suggests how non-natural operators may be redefined by OSOS rules.

Example 20 Let an operator g be defined by a single rule r below.

$$\frac{X \stackrel{a}{\rightarrow} Y \quad X \stackrel{b}{\not\rightarrow}}{g(X) \stackrel{a}{\rightarrow} t}$$

Clearly, g is not natural as there is no rule for g with the premises $\{X \stackrel{b}{\rightarrow} Y\}$. We see no other way to redefine g but to use some kind of auxiliary rules. We give three different definitions of g in terms of OSOS rules. First, consider the rules

$$\frac{X \stackrel{a}{\rightarrow} Y}{g(X) \stackrel{a}{\rightarrow} t}\ r_a \qquad\qquad \frac{X \stackrel{b}{\rightarrow} Y}{g(X) \stackrel{b}{\rightarrow} \mathbf{0}}\ r_b$$

which are ordered by $r_a < r_b$. If $p \stackrel{b}{\not\rightarrow}$ and $p \stackrel{a}{\rightarrow}$, then it is clear that $g(p) \stackrel{a}{\rightarrow}$ by rule r_a. But, when $p \stackrel{b}{\rightarrow}$, we obtain $g(p) \stackrel{b}{\rightarrow}$ by r_b. This transition cannot be derived by r. To stop this, we impose $r_b < r_b$ which makes r_b never enabled at $g(p)$.

The second alternative definition of g uses an auxiliary restriction operator instead of auxiliary rules that are above themselves and thus never enabled. Consider rule r_a above and the following two rules, where *error* is a new action:

$$\frac{X \xrightarrow{b} Y}{g(X) \xrightarrow{error} \mathbf{0}}\ r_{error} \qquad\qquad \frac{X \xrightarrow{a} Y}{X \setminus error \xrightarrow{a} Y \setminus error}\ \text{if } a \neq error$$

The ordering $<$ satisfies $r_a < r_{error}$. It can be easily checked that the process $g(p) \setminus error$ has the same behaviour as the original process $g(p)$.

The third way to redefine g is by the following rules:

$$\frac{X \xrightarrow{a} Y}{g(X) \xrightarrow{a} t}\ r_a \qquad\qquad \frac{X \xrightarrow{a} Y \quad X \xrightarrow{b} Y'}{g(X) \xrightarrow{b} \mathbf{0}}\ r'_b$$

The ordering satisfies $r_a < r'_b$ and $r'_b < r_a$. Notice that when $p \stackrel{b}{\nrightarrow}$ and $p \xrightarrow{a}$ then r_a is applicable because higher priority r'_b is not applicable. On the other hand, r'_b is never applicable: if $p \xrightarrow{b}$ and $p \xrightarrow{a}$, then r'_b is not applicable because higher priority r_a is applicable. As a result, $g(p) \xrightarrow{a}$ if $p \stackrel{b}{\nrightarrow}$ and $p \xrightarrow{a}$.

Now, we are ready to state our main expressiveness result.

Theorem 21 (GSOS = OSOS) Any GSOS language can be equivalently given as an OSOS language, and conversely.

Proof. We show that any GSOS operator can be turned into an OSOS operator, and that any OSOS operator can be turned into a GSOS operator. We fix a signature Σ and a set of actions A.

First, suppose that $f \in \Sigma$ is an n-ary GSOS operator with set of rules R_f. We turn f into an OSOS operator by the method used in the first of the alternative definitions for operator g in Example 20. The set R'_f of OSOS rules for f is

$$R'_f = R^{\dagger}_f \cup R^{\mathrm{aux}}_f,$$

where $R^{\dagger}_f$ and R^{aux}_f are defined as follows, and $a \in A$ is any action:

$$R^{\dagger}_f \;= \{r^{\dagger} \mid r \in R_f\},$$

$$R^{\mathrm{aux}}_f = \{\ \frac{X_i \xrightarrow{\alpha} X'_i}{f(\boldsymbol{X}) \xrightarrow{a} \mathbf{0}} \mid \exists r \in R_f.\ X_i \stackrel{\alpha}{\nrightarrow} \in pre(r) \ \wedge\ \forall r' \in R_f.\ \neg(r \stackrel{X_i\,\alpha}{\prec} r')\}.$$

Notice that we could have defined R_f^{aux} simply as

$$\{ \frac{X_i \xrightarrow{\alpha} X_i'}{f(\boldsymbol{X}) \xrightarrow{a} \mathbf{0}} \mid \exists r \in R_f.\ X_i \stackrel{\alpha}{\nrightarrow} \in \mathit{pre}(r)\}$$

but for the sake of efficiency (see remark below) we prefer the smaller set.

Notice that if f is natural, then R_f^{aux} is empty. This is because, by Definition 18, for each rule $r \in \mathit{rules}(f)$ with $X_i \stackrel{\alpha}{\nrightarrow} \in \mathit{pre}(r)$, there exists $r' \in \mathit{rules}(f)$ such that $r \stackrel{X_i\ \alpha}{\prec} r'$.

By the definition of R_f' we know that, for each negative rule $r \in R_f$ and for each premise $X_i \stackrel{\alpha}{\nrightarrow}$ of r, there is a rule $r' \in R_f'$ with the single premise $X_i \xrightarrow{\alpha} X_i'$ for some X_i'.

The ordering $<$ on rules in R_f' is defined as

$$<_f\ = \{(r^\dagger, r'^\dagger) \mid r, r' \in R_f \cup R_f^{\mathrm{aux}} \ \wedge\ \exists X, \alpha.\ r \stackrel{X\ \alpha}{\prec} r'\} \ \cup\ \{(r, r) \mid r \in R_f^{\mathrm{aux}}\}.$$

Notice that since, for each $r \in R_f^{\mathrm{aux}}$, we have $r <_f r$, all such rules r are never enabled.

If f is natural, then no rule in R_f' is above itself. This is because (a) R_f^{aux} is empty and (b) rules in R_f have no contradictory premises, so $\neg(r \stackrel{X\ \alpha}{\prec} r)$ holds for all $r \in R_f, X$ and α.

Remark. If a GSOS operator has any negative rules, it is possible that its OSOS version has more rules. However, as far as the amount of computation required to derive transitions is concerned, which is measured by the total number of transitions and negative transitions that need to be checked, it is easy to verify that this amount is the same for both classes of process languages.

Remark. Although our translations of GSOS operators into OSOS operators may be considered somewhat artificial, for example because of the use of auxiliary rules which are above themselves and thus never enabled, in many cases they are simple and intuitive. For example, the GSOS versions of sequential composition and priority operators (see Section 6), both of which are natural operators, can be defined without auxiliary rules.

For the converse direction of the proof, suppose that f is an n-ary OSOS operator with set of ordered rules R_f. For $r \in R_f$ we show how to construct the set $R_f'(r)$ of GSOS rules for f which corresponds to $\{r\} \cup \mathit{higher}(r)$. Then, the set R_f' of GSOS rules for f is $\bigcup_{r \in R_f} R_f'(r)$. If $\mathit{higher}(r) = \emptyset$, then clearly $R_f'(r) = \{r\}$. Otherwise, assume $\mathit{higher}(r) = \{r_i \mid 1 \le i \le m\}$. If one of the rules in $\mathit{higher}(r)$, say r', is an axiom, then by Definition 9 rule r is never enabled because r' is always applicable. Thus, $R'_f(r) = \emptyset$. So, assume that none of the rules in $\mathit{higher}(r)$ is an axiom. Then,

$$R'_f(r) = \{ \frac{\mathit{pre}(r) \cup \{\neg T_1, \ldots, \neg T_m\}}{\mathit{con}(r)} \mid \bigwedge_{i\in\{1,\ldots,m\}} T_i \in \mathit{pre}(r_i)\}.$$

This says that the set of negative premises in any rule in $R'_f(r)$ consists of the negations of m transitions $\{\neg T_1, \ldots, \neg T_m\}$, where each T_i is one of the (positive) premises in $r_i \in \mathit{higher}(r)$.
Remark. We easily calculate that $R'_f(r)$ has $\prod_{i=1}^{m} k_i$ rules. Thus, when m and some k_i are greater than 1 it is clear that the fragment of the definition for f consisting of r and $\mathit{higher}(r)$ is more concise than the corresponding fragment $R'_f(r)$. $\square$

It is natural to ask whether the OSOS format has the same expressive power if we require that the ordering relation is a *partial ordering* (the so-called *partial* OSOS format). We show that partial OSOS is just as expressive, by showing that we can translate GSOS into partial OSOS.

Definition 22 A process language is *strict* OSOS if it is OSOS and the ordering of its rules is irreflexive. A process language is *partial* OSOS if it is OSOS and the ordering of its rules is irreflexive and transitive.

First, we note the following:

Proposition 23 Any natural GSOS language can be equivalently given as a strict OSOS language.

Proof. We show that any natural GSOS operator f can be turned into a strict OSOS operator. We use the ordering constructed in the proof of Theorem 21. We need only show that this ordering is irreflexive. Firstly, notice that, since f is natural, R_f^{aux} is empty. This is because by Definition 18, for every rule for f with a negative premise there is a rule for f with a sole positive premise corresponding to the negative premise. Secondly, since rules for natural GSOS operators have no contradictory premises we have $\neg(r \overset{X\,\alpha}{\prec} r)$ for all rules r for f and all X and α. Therefore, the ordering is irreflexive. Additionally, as can be seen in Example 19, the orderings are not in general transitive. $\square$

Proposition 24 Any GSOS language can be equivalently given as a partial OSOS language.

Proof. Assume a GSOS language $L = (\Sigma, A, \rightarrow)$. We show that any GSOS operator $f \in \Sigma$ can be turned into an equivalent partial OSOS term. The proof generalises the second translation method for g in Example 20. Let *error* be a new action (not in A), and let $\backslash error$ be the restriction operator of Example 20, which restricts the action *error*. Let f have set of rules R_f. The set R'_f of OSOS rules for f is $R^{\dagger}_f \cup R^{\mathrm{aux}}_f$, where

$$R^{\mathrm{aux}}_f = \{ \frac{X_i \xrightarrow{\alpha} X'_i}{f(\boldsymbol{X}) \xrightarrow{error} \mathbf{0}} \mid \exists r \in R_f.\ X_i \stackrel{\alpha}{\nrightarrow} \in pre(r) \}.$$

The ordering on R'_f is $<_f = \{(r^{\dagger}, r') \mid r \in R_f \ \wedge\ r' \in R^{\mathrm{aux}}_f \ \wedge\ \exists X, \alpha.\ r \stackrel{X\,\alpha}{\prec} r'\}$. We easily check that $<_f$ is irreflexive; it relates only some rules in $R^{\dagger}_f$ to some rules in R^{aux}_f, and $R^{\dagger}_f \cap R^{\mathrm{aux}}_f = \emptyset$. For the same reason $<$ is transitive. Notice that some terms $f(\boldsymbol{p})$ in the new OSOS language may have extra *error* transitions because some of the rules in R^{aux}_f may be applicable. We stop *error* transitions with the restriction operator $\backslash error$. Thus $f(\boldsymbol{X})$ is translated to $f(\boldsymbol{X})\backslash error$. $\Box$

Proposition 24 appears to subsume Proposition 23, but if we look at the proofs, we see that in the latter case we were forced to give f some spurious *error* transitions, and to introduce a new operator to remove them. Instead of a "signature" translation, taking operators to operators (as in Theorem 21 and Proposition 23), we gave a "compositional" translation in Proposition 24, taking operators to terms. Such compositional embeddings have been mentioned in [16].

4 Process Languages for Branching and Eager Bisimulations

In this section we show how rules with silent actions can be safely used, together with GSOS and OSOS rules, in defining process operators which preserve branching and eager bisimulations. Our aim is to express the character of silent actions, as being unobservable and independent of the environment, through several conditions on the form of rules with silent actions and how they can be combined with other ordered or unordered rules. We also identify the forms of copies of process variables in rules that are 'safe' for eager and branching bisimulation, i.e. operators defined by rules with these forms of copies preserve the relevant preorders.

We define several classes of process languages and prove that eager and branching bisimulation preorders are preserved by operators in the relevant classes.

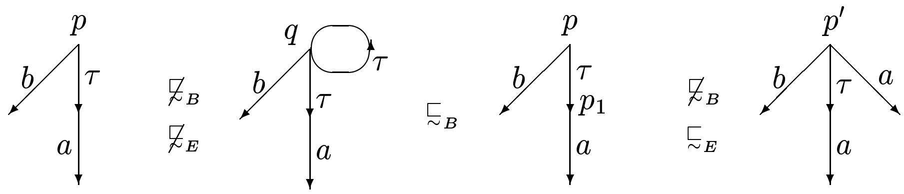

Fig. 1.

4.1 Branching and Eager Bisimulations

We recall the definitions of branching and eager bisimulation relations. Before we give the definitions, we restate some standard notation.

Let p, p' and q be processes and α and τ be actions over an LTS $(\mathcal{P}, A, \rightarrow)$. We write $p \xrightarrow{\alpha}$ when there is q such that $p \xrightarrow{\alpha} q$, and $p \not\xrightarrow{\alpha}$ otherwise. Expression $p \xRightarrow{\tau} q$ denotes $p(\xrightarrow{\tau})^* q$. We say that q is a τ-*derivative* of p if $p \xRightarrow{\tau} q$. $p\Uparrow$ is read as "p is *divergent*" and it means $p(\xrightarrow{\tau})^\omega$, i.e. p can perform an infinite sequence of silent actions. We say "p is *convergent*", written as $p\Downarrow$, if p is not divergent. Finally, if $\alpha = \tau$, then $p \xrightarrow{\widehat{\alpha}} p'$ means $p \xrightarrow{\tau} p'$ or $p = p'$; otherwise it is simply $p \xrightarrow{\alpha} p'$.

Definition 25 Assume an LTS $(\mathcal{P}, A, \rightarrow)$. A relation $R \subseteq \mathcal{P} \times \mathcal{P}$ is a *branching bisimulation* if whenever pRq, then properties (a), (b) and (c) below hold for all appropriate α, p' and q'.

$$(a) \qquad p \xrightarrow{\alpha} p' \text{ implies } \exists q', q''.(q \xRightarrow{\tau} q' \xrightarrow{\widehat{\alpha}} q'' \text{ and } pRq' \text{ and } p'Rq'')$$

$$(b) \qquad p\Downarrow \text{ implies } q\Downarrow$$

$$(c) \quad p\Downarrow \text{ and } q \xrightarrow{\alpha} q' \text{ implies } \exists p', p''.(p \xRightarrow{\tau} p' \xrightarrow{\widehat{\alpha}} p'' \text{ and } p'Rq \text{ and } p''Rq')$$

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

A relation $R \subseteq \mathcal{P} \times \mathcal{P}$ is an *eager bisimulation* if R is defined as branching bisimulation but without pRq' in (a) and without $p'Rq$ in (c). $p \underset{\sim}{\sqsubset}_E q$ if there exists an eager bisimulation R such that pRq.

Example 26 We show that branching bisimulation is strictly finer than eager bisimulation, i.e. $\underset{\sim}{\sqsubset}_B \subsetneq \underset{\sim}{\sqsubset}_E$. Consider the processes in Figure 1. The $\subseteq$ inclusion follows from the definition. The counterexample for equality are processes p and p'. We have $p \underset{\sim}{\sqsubset}_E p'$ but *not* $p \underset{\sim}{\sqsubset}_B p'$, for although $p' \xrightarrow{a}$ and $p \xRightarrow{\tau} p_1 \xrightarrow{a}$ but clearly not $p' \underset{\sim}{\sqsubset}_B p_1$.

Moreover, we show that eager bisimulation is strictly finer than the standard weak bisimulation $\approx$ as defined in [27]. Processes r and s in Figure 2 are weak bisimilar by the Milner's third τ-law [27] but they are not eager bisimilar. This is because after $s \xrightarrow{a} s'$ there is no r' such that $r \xRightarrow{\tau}\xrightarrow{a} r'$ with $r' \underset{\sim}{\sqsubset}_E s'$.

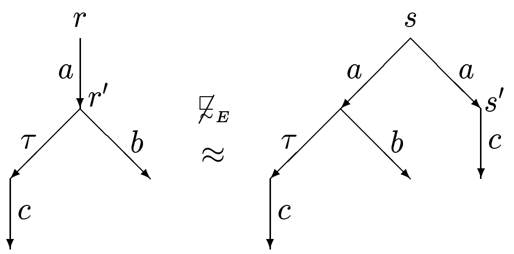

Fig. 2.

It is clear that $\underset{\approx}{\sqsubset}_B$ and $\underset{\approx}{\sqsubset}_E$ are preorders. Our branching bisimulation relation is a possible generalisation of the standard notion defined in [19,20]. We make the relation sensitive to divergence in the same way as was done with weak bisimulation in [26,1,48]. Moreover, we have simplified the definition of branching bisimulation by not including expressions $q'' \overset{\tau}{\Rightarrow} q'''$ and $p'Rq'''$ inside the ( ) parenthesis in property (a) and $p'' \overset{\tau}{\Rightarrow} p'''$ and $p'''Rq'$ inside the ( ) parenthesis in (c). It can be easily checked that branching bisimulation as defined in Definition 25 and a relation that satisfies properties (a), (b) and (c), with (a) and (c) augmented to include the additional expressions described above, are in fact equal as relations.

Preorder $\underset{\approx}{\sqsubset}_E$ is the version of weak bisimulation studied in [26,1,48,41,43,44], where testing, modal logic and axiomatic characterisations were proposed and congruence results with respect to the ISOS and wb formats were proved. $\underset{\approx}{\sqsubset}_E$ coincides with a *delay* bisimulation [50,20] for processes with no divergence. We have chosen this finer version of weak bisimulation in preference to the standard weak bisimulation [27] because formats for eager bisimulation [40,41,43] are considerably simpler than those for the standard weak bisimulation [13]. Also, since we take divergence into account, formats for eager bisimulation may have rules with negative premises [40,41] and thus are more general than those for weak bisimulation [13]. Moreover, unlike eager bisimulation, weak bisimulation is not preserved by some simple but useful operators (and the problem is not due to the initial silent actions). For example, the action refinement operator defined in Section 6 preserves eager bisimulation but not weak bisimulation. Having said that, we agree that both bisimulation relations are equally suitable for many process languages that are defined by rules with no negative premises and where divergence is not considered.

4.2 bb *and* eb *Languages*

In this subsection we introduce formats of unordered rules, and thus classes of process languages, which preserve branching and eager bisimulation preorders. Our formats will consist of positive GSOS rules that satisfy several conditions defining permissible use of silent actions and copies of process variables (resources).

Notation: In order to shorten the presentation of conditions on rules and orderings in this and the next subsection we will leave out the outermost universal quantifiers binding $f \in \Sigma$ and $r, r' \in \mathit{rules}(f)$, where appropriate.

The first attempt to give a uniform treatment to silent actions in formats of rules is due to Bloom [12]. He introduced τ-rules, which are also used in [46,40]. In our opinion τ-rules embody the independent of the environment and uncontrollable character of τ actions: 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 the contributing arguments are the active arguments, so the first condition is the following:

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

We can show that process operators which do not satisfy this condition, for example the CCS choice operator $+$ and the ACP left-merge, are not well behaved: they do not preserve many weak equivalences. The τ-rules for any operator f that are guaranteed by condition (1) shall be henceforth called the τ-rules *associated with* f. Similarly, given any rule r for f, all τ-rules τ_i for f such that $i \in \mathit{active}(r)$ shall be called the τ-rules *associated with* r.

The character of silent actions is not fully represented by insisting only that all operators have their associated τ-rules. We additionally insist that silent actions are unobservable: silent actions of subprocesses cannot produce visible actions of the process or change its structure [40]. This principle can be formulated as follows:

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

The condition requires that no rules except τ-rules can have τ actions in the premises. As a result, it disallows troublesome operators like the *see*-τ, defined below, which can count silent actions. We easily check that *see*-$\tau(X)$ distinguishes weakly trace equivalent processes $\tau.\mathbf{0}$ and $\tau.\tau.\mathbf{0}$.

$$\frac{X \xrightarrow{\tau} X'}{\mathit{see}\text{-}\tau(X) \xrightarrow{a} \mathit{see}\text{-}\tau(X)}$$

Next, we develop conditions which describe several ways in which process operators use their process resources, and we investigate which of these ways are 'safe' for eager and branching bisimulations.

Results in [41] show that positive ISOS operators preserve eager bisimulation. We recall that positive ISOS operators are those operators which (a) are definable by positive GSOS rules that satisfy the above two conditions, and (b) use

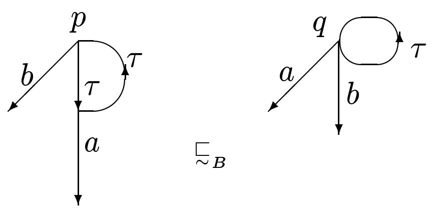

Fig. 3.

their process resources *linearly*. In the setting of transition rules, the argument X_i is used linearly in a rule r if and only if whenever $X_i \xrightarrow{\alpha} Y_i$ is a premise of r, then X_i appears in no other premise of r and it does not appear in the target of r. In other words, a linear use of process resources is synonymous with no implicit copies in rules, or equivalently with only explicit copies in rules.

In order to make the presentation self-contained we recall the definition of the ISOS format [41].

Definition 27 A set of rules is in the ISOS format if it consists of ISOS rules, defined below, and their associated τ-rules. An ISOS rule is an expression with the form

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

where $\boldsymbol{Y}$ is the sequence of variables $Y_1,\ldots,Y_n$ and $Y_j = X_j$ if $j \notin I$, and variables X_i, Y_i are all distinct. Moreover, $I, K \subseteq \{1,\ldots,n\}$, all L_k are finite subsets of $\mathbb{N}$, and $C[\boldsymbol{Y}]$ is a context. Expressions $X_k \stackrel{\tau}{\not\to}\stackrel{b_{kl}}{\not\to}$ denote pairs of negative transitions $X_k \stackrel{\tau}{\not\to}$ and $X_k \stackrel{b_{kl}}{\not\to}$; they are called *refusal* transitions.

It is obvious that the ISOS format is a subformat of the GSOS format. Moreover, we easily check that positive ISOS rules (i.e. ISOS rules without refusal transitions) can only have explicit copies of process variables but not implicit copies. Positive GSOS rules, however, can have both explicit and implicit copies of process variables. As a result, not all operators which are definable by positive GSOS rules satisfying conditions (1) and (2) preserve eager and branching bisimulations. This problem is illustrated as follows: Consider the operators *a-and-b*, *a-then-b* and *then-b* from [40,41]. They are defined by the following rules together with their associated τ-rules (which are not shown):

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

We easily verify that the defining rules are positive GSOS and that they satisfy conditions (1) and (2). According to Definition 4, the first rule has implicit

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

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

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

$$\mathit{implicit\text{-}copies}(r) = \emptyset \tag{4}$$

Fig. 4. Conditions for branching bisimulation and eager bisimulation rules.

copies of X in the premises and the second rule has implicit copies of X in the target. Consider eager bisimilar processes p and p' in Figure 1. One can easily check that $a\text{-}and\text{-}b(p) \not\stackrel{c}{\Rightarrow}$ but $a\text{-}and\text{-}b(p') \stackrel{c}{\Rightarrow}$. Also, $a\text{-}then\text{-}b(p) \not\stackrel{c}{\Rightarrow}$ but $a\text{-}then\text{-}b(p') \stackrel{c}{\Rightarrow}$. Thus, the operators $a\text{-}and\text{-}b$ and $a\text{-}then\text{-}b$ can distinguish between some eager bisimilar processes. Hence, operators defined by rules with implicit copying may not preserve eager bisimulation.

It is also the case that the operator $a\text{-}and\text{-}b$ can distinguish between some branching bisimilar processes. Consider processes p and q in Figure 3. Clearly, they are branching bisimilar and also divergent. We have $a\text{-}and\text{-}b(q) \stackrel{c}{\Rightarrow}$, but $a\text{-}and\text{-}b(p) \not\stackrel{c}{\Rightarrow}$ because p cannot silently reach a state where both a and b are immediately available. Hence, operators defined by rules with implicit copying in the premises may not preserve branching bisimulation. However, we claim that implicit copies in the target are 'safe' for branching bisimulation.

The above discussion concerning the use of process resources suggests the following conditions:

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

$$\mathit{implicit\text{-}copies}(r) = \emptyset \tag{4}$$

The conditions for branching and eager bisimulation rules are gathered for convenience in Figure 4.

We are ready to define formats of unordered rules, and thus classes of process languages, which preserve branching and eager bisimulation preorders.

Definition 28 A set of positive GSOS rules is called bb, or branching bisimulation, if its rules satisfy conditions (1), (2) and (3). A set of positive GSOS rules is called eb, or eager bisimulation, if its rules satisfy conditions (1), (2) and (4). The bb (eb) format consists of bb (eb) rules. An operator is bb (eb) if it is defined by bb (eb) rules.

This definition together with Definition 16 gives rise to bb and eb process languages, and the classes of such languages.

We note that bb rules allow explicit copies and implicit copies in the target but eb rules allow only explicit copies. It is easy to check that the eb format coincides with the positive ISOS format and the bb format is an extension of the positive ISOS format with implicit copies in the target.

Most of the popular process operators are eb, and thus bb, but there are important exceptions, for example the CCS $+$, the ACP left-merge and the Kleene star operator [6,4,17].

Finally, we are ready to state and prove the main results of this subsection. Firstly, we give alternative characterisations of branching bisimulation and eager bisimulation preorders that will be essential in the proofs of congruence results in this subsection.

Proposition 29 Given the LTS $(\mathrm{T}(\Sigma), \mathsf{Act}, \to)$, a relation $B \subseteq \mathrm{T}(\Sigma) \times \mathrm{T}(\Sigma)$ is a branching bisimulation if and only if, for all p and q such that pBq, the following properties hold for all appropriate α, p' and q':

$$(B.a) \qquad p \xrightarrow{\alpha} p' \ \text{ implies } \ \exists q', q''.(q \stackrel{\tau}{\Rightarrow} q' \xrightarrow{\widehat{\alpha}} q'' \ \text{ and } \ pBq' \ \text{ and } \ p'Bq')$$

$$(B.b) \qquad p \stackrel{\tau}{\not\to} \ \text{ implies } \ q\Downarrow$$

$$(B.c) \quad p\Downarrow \text{ and } q \xrightarrow{\alpha} q' \ \text{ implies } \ \exists p', p''.(p \stackrel{\tau}{\Rightarrow} p' \xrightarrow{\widehat{\alpha}} p'' \ \text{ and } \ p'Bq \ \text{ and } \ p''Bq')$$

A relation $B \subseteq \mathrm{T}(\Sigma) \times \mathrm{T}(\Sigma)$ is an eager bisimulation if and only if, for all p and q such that pBq, the above properties but without pBq' and $p'Bq$ hold.

Theorem 30 The bb class preserves branching bisimulation preorder.

Proof. Let $G = (\Sigma, \mathsf{Act}, \to)$ be any bb process language. We start by defining a relation $\mathcal{B} \subseteq \mathrm{T}(\Sigma) \times \mathrm{T}(\Sigma)$ as the least relation that satisfies $\boldsymbol{u}\mathcal{B}\boldsymbol{v}$ if $\boldsymbol{u} \sqsubseteq_{B} \boldsymbol{v}$, and $C[\boldsymbol{u}]\mathcal{B}C[\boldsymbol{v}]$ if $\boldsymbol{u}\mathcal{B}\boldsymbol{v}$, where $C[\boldsymbol{X}]$ is a Σ context and $\boldsymbol{u}$ and $\boldsymbol{v}$ are vectors of closed Σ terms of appropriate length. Hence, $\mathcal{B}$ is the least congruence which contains $\sqsubseteq_{B}$. All we need to do is to show that $\mathcal{B}$ is a branching bisimulation relation. We will show this by proving properties $(B.a)$, $(B.b)$ and $(B.c)$ of Proposition 29. Then, we would have $\mathcal{B} = \sqsubseteq_{B}$ and since $\mathcal{B}$ is preserved by all Σ contexts so $\sqsubseteq_{B}$ would be preserved too. Our proof is in two parts:

- Firstly, we show by structural induction that $\mathcal{B}$ satisfies $(B.a)$ and $(B.c)$;
- Then, we show that $\mathcal{B}$, which satisfies $(B.a)$ and $(B.c)$, also satisfies $(B.b)$.

First, we re-examine the structure of process terms related by $\mathcal{B}$. By the definition of $\mathcal{B}$, $p\mathcal{B}q$ implies that either $p \sqsubseteq_{B} q$ or there is a non-trivial context $C[\boldsymbol{X}]$ and appropriate vectors of process terms $\boldsymbol{p}$ and $\boldsymbol{q}$ such that $p = C[\boldsymbol{p}]$ and $q = C[\boldsymbol{q}]$ with $\boldsymbol{p}\mathcal{B}\boldsymbol{q}$. In the first case, p and q obviously satisfy $(B.a)$,

$(B.b)$ and $(B.c)$. In the latter case, p and q can also be expressed as $f(\boldsymbol{u})$ and $f(\boldsymbol{v})$ respectively, for some $f \in \Sigma$ and appropriate $\boldsymbol{u}$ and $\boldsymbol{v}$. Moreover, all the corresponding elements u_i and v_i, of $\boldsymbol{u}$ and $\boldsymbol{v}$ respectively, either satisfy $u_i \sqsubseteq_{\approx B} v_i$ or $u_i = C_i[\boldsymbol{p}]$ and $v_i = C_i[\boldsymbol{q}]$, for some context $C_i[\boldsymbol{X}]$, and thus $u_i \mathcal{B} v_i$. In both cases we clearly have $\boldsymbol{u}\mathcal{B}\boldsymbol{v}$.

We begin the proof of $(B.a)$ and $(B.c)$. Let $p\mathcal{B}q$, and assume that $(B.a)$ and $(B.c)$ hold for all subterms of p and q that are related by $\mathcal{B}$. If $p \sqsubseteq_{\approx B} q$, then we are done. Otherwise, as explained above, p and q can be represented as $f(\boldsymbol{u})$ and $f(\boldsymbol{v})$ respectively, where $f \in \Sigma_n$, $\boldsymbol{u}$ and $\boldsymbol{v}$ are of length n and $\boldsymbol{u}\mathcal{B}\boldsymbol{v}$.

(1) For $(B.a)$ we prove that in fact the following stronger property holds for $f(\boldsymbol{u})$ and $f(\boldsymbol{v})$, and for all μ-derivatives u' of $f(\boldsymbol{u})$.

$$\begin{aligned} f(\boldsymbol{u}) \xrightarrow{\mu} u' \ \text{ implies } \ \exists D[\boldsymbol{X},\boldsymbol{Y}], \boldsymbol{u}', \boldsymbol{v}', \boldsymbol{v}''. \ (f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} f(\boldsymbol{v}') \xrightarrow{\widehat{\mu}} D[\boldsymbol{v}', \boldsymbol{v}''] \\ \text{and } u' = D[\boldsymbol{u}, \boldsymbol{u}'] \\ \text{and } \ \boldsymbol{u}\mathcal{B}\boldsymbol{v}' \text{ and } \ \boldsymbol{u}'\mathcal{B}\boldsymbol{v}'') \end{aligned}$$

In turn, this property together with the definition of $\mathcal{B}$ implies $(B.a)$. Assume $f(\boldsymbol{u}) \xrightarrow{\mu} u'$. This transition can be either derived by an action rule or by a τ-rule. We only consider the first case as the second case follows similarly. Let $f(\boldsymbol{u}) \xrightarrow{\mu} u'$ be derived by the following positive GSOS rule r, which in accordance with condition (3) has no implicit copies of process variables in the premises (but may have implicit copies in the target), using a ground substitution ρ defined by $\rho(\boldsymbol{X}) = \boldsymbol{u}$ and $\rho(Y_i) = u_i'$ for all $i \in I$.

$$\frac{\{X_i \xrightarrow{a_i} Y_i\}_{i \in I}}{f(\boldsymbol{X}) \xrightarrow{\mu} E[\boldsymbol{X}, \boldsymbol{Y}]}$$

The conclusion of r is valid under ρ in $\rightarrow$, i.e. $(f(\boldsymbol{u}), \mu, \rho(E[\boldsymbol{X}, \boldsymbol{Y}])) \in \rightarrow$ or equivalently $f(\boldsymbol{u}) \xrightarrow{\mu} \rho(E[\boldsymbol{X}, \boldsymbol{Y}])$.

Let $\boldsymbol{u}'$ denote the sequence of all u_i' so that $\rho(\boldsymbol{Y}) = \boldsymbol{u}'$. Hence, the required context $D[\boldsymbol{X}, \boldsymbol{Y}]$ is simply $E[\boldsymbol{X}, \boldsymbol{Y}]$ and $u' = E[\boldsymbol{u}, \boldsymbol{u}']$. Also, the premises of r are valid under ρ in $\rightarrow$, i.e. $(\rho(X_i), a_i, \rho(Y_i)) \in \rightarrow$ or equivalently $u_i \xrightarrow{a_i} u_i'$ for all $i \in I$.

Next, we employ the inductive hypothesis for each u_i and v_i with $i \in I$. Since $(B.a)$ holds for each u_i and v_i we deduce $v_i \stackrel{\tau}{\Rightarrow} v_i' \xrightarrow{a_i} v_i''$ and $u_i \mathcal{B} v_i'$ with $u_i' \mathcal{B} v_i''$. Let $\boldsymbol{v}'$ stand for the sequence $v_1', \ldots, v_n'$ such that $v_k' = v_k$ when $k \notin I$. Moreover, let $\boldsymbol{v}''$ denote the sequence of all v_i''. Hence, we obtain $\boldsymbol{u}\mathcal{B}\boldsymbol{v}'$ and $\boldsymbol{u}'\mathcal{B}\boldsymbol{v}''$ as required.

Finally, we need to show $f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} f(\boldsymbol{v}')$, and then $f(\boldsymbol{v}') \xrightarrow{\mu} v''$ for appropriate v''. $f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} f(\boldsymbol{v}')$ is obtained by combining transitions derived by the appropriate τ-rules applied to each $v_i \stackrel{\tau}{\Rightarrow} v_i'$ with $i \in I$. $f(\boldsymbol{v}') \xrightarrow{\mu} v''$

is obtained by rule r with the substitution ρ' defined by $\rho'(\boldsymbol{X}) = \boldsymbol{v}'$ and $\rho'(Y_i) = v''_i$ for all appropriate i. This is possible because the premises of r are valid under ρ' in $\rightarrow$, namely $v'_i \xrightarrow{a_i} v''_i$ for all i. Thus, with $\boldsymbol{v}'$ and $\boldsymbol{v}''$ as above, we obtain $v'' = E[\boldsymbol{v}', \boldsymbol{v}'']$.

(2) The proof of property $(R.c)$ is very similar to the above proof of property $(R.a)$. The only difference is that in order to use the $(R.c)$ part of the inductive hypothesis we need to establish that if $f(\boldsymbol{u})\Downarrow$, then $u_i\Downarrow$ for all $i \in \mathit{active}(f)$. However, this follows by the τ-rules associated with f.

So far we have shown that processes related by $\mathcal{B}$ satisfy properties $(B.a)$ and $(B.c)$. For property $(B.b)$ we need to show that if $p\mathcal{B}q$ and $p \stackrel{\tau}{\nrightarrow}$, then $q\Downarrow$. We proceed by structural induction and by a proof by contradiction. If $p \sqsubseteq\!\!\!\sim_B q$, then we are done. Otherwise, assume $p = f(\boldsymbol{u})$ and $q = f(\boldsymbol{v})$, thus $\boldsymbol{u}\mathcal{B}\boldsymbol{v}$. Let $f(\boldsymbol{u}) \stackrel{\tau}{\nrightarrow}$. Also, assume that $(B.b)$ holds for all pairs of processes u_i and v_i, and $(B.c)$ holds for $f(\boldsymbol{u})$ and $f(\boldsymbol{v})$.

Assume for a contradiction $f(\boldsymbol{v})\Uparrow$. It implies, by the definition of the divergence predicate, that there exist process terms q_i, for $1 \leq i < \omega$, such that

$$f(\boldsymbol{v}) \xrightarrow{\tau} q_1 \xrightarrow{\tau} \cdots q_i \xrightarrow{\tau} \cdots.$$

Since $(B.c)$ holds for $f(\boldsymbol{u})$ and $f(\boldsymbol{v})$, and since $f(\boldsymbol{u})$ is stable, i.e. $f(\boldsymbol{u}) \stackrel{\tau}{\nrightarrow}$, the above sequence can be matched by

$$f(\boldsymbol{u}) \stackrel{\tau}{\Rightarrow} f(\boldsymbol{u}) \stackrel{\tau}{\Rightarrow} \cdots f(\boldsymbol{u}) \stackrel{\tau}{\Rightarrow} \cdots,$$

where $f(\boldsymbol{u})\mathcal{B}q_i$. In general, any τ transition in the first sequence above can be derived either by an action rule or by a τ-rule. We can easily show by induction that every τ transition in the first sequence above is only derivable by a τ-rule. The argument is as follows: If any τ transition in the first sequence can be derived by an action rule, then we can deduce $f(\boldsymbol{u}) \xrightarrow{\tau}$ by $(B.c)$, thus contradicting the stability of $f(\boldsymbol{u})$. Therefore, the first sequence is in fact

$$f(\boldsymbol{v}) \xrightarrow{\tau} f(\boldsymbol{v}_1) \xrightarrow{\tau} \cdots f(\boldsymbol{v}_i) \xrightarrow{\tau} \cdots$$

and $\boldsymbol{v}_i \xrightarrow{\tau} \boldsymbol{v}_{i+1}$, for all i, where the notation $\boldsymbol{v}_l \xrightarrow{\tau} \boldsymbol{v}_{l+1}$ means $v_{lk} \xrightarrow{\tau} v_{l+1k}$, for one particular k, and $v_{lj} = v_{l+1j}$ for all $j \neq k$. Since f has only finitely many arguments we deduce that one of the elements in $\boldsymbol{v}$ diverges. Let this term be v_d, so $v_d\Uparrow$. Let u_d be the corresponding element in $\boldsymbol{u}$. Then, since $u_d\mathcal{B}v_d$ and $u_d \stackrel{\tau}{\nrightarrow}$ we obtain $v_d\Downarrow$ by the inductive hypothesis. This contradicts $v_d\Uparrow$, so we obtain that $f(\boldsymbol{v})\Downarrow$ as required. □

Theorem 31 The eb class preserves eager bisimulation preorder.

Proof. The proof follows very much the same pattern as the proof of Theorem 30 using Proposition 29. Alternatively, since the eb format coincides with the positive ISOS format [40,41] the above result is a corollary of the congruence theorem in [41] for the ISOS format and the version of weak bisimulation which we call here eager bisimulation. □

4.3 bbo *and* ebo *Languages*

In this subsection we consider bb and eb rules equipped with orderings. We propose formats of such ordered rules, and thus classes of process languages, which preserve branching and eager bisimulation preorders.

A careless ordering on rules may change the unobservable and independent of the environment character of silent actions. We give several examples which illustrate this problem and propose a number of conditions on orderings which safeguard the traditional character of silent actions.

Consider a version of the interleaved parallel composition operator '$\|$' defined by the rule schemas below, where a is any action in Vis, together with τ-rules τ_1 and τ_2 which are not shown.

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

The ordering is defined by $\tau_2 < r_{a*}$. Consider trace equivalent processes $a.b.\mathbf{0}$ and $a.\tau.b.\mathbf{0}$. It is easy to see that these processes can be distinguished by $\|$: we have $c.\mathbf{0} \parallel a.b.\mathbf{0} \xrightarrow{ab}$ but $c.\mathbf{0} \parallel a.\tau.b.\mathbf{0} \xrightarrow{a} c.\mathbf{0} \parallel \tau.b.\mathbf{0} \not\xrightarrow{\tau}$ since $\tau_2 < r_{c*}$. Thus, the first condition might be as follows:

$$\text{if } r \text{ is a } \tau\text{-rule then } \mathit{higher}(r) = \emptyset$$

The intuition behind this condition is that τ-rules should not have lower priority. But, although the condition is intuitive 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 only, as in the case of the sequential composition operator. This may be defined by placing some rules with the first argument in the premises above τ_2. We allow such orderings provided that all rules which are above τ_2 are also above all the rules with active second argument. A better candidate for the first condition might be as follows:

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

When orderings are transitive relations, as in the preliminary version of this work [43], the above condition is what we need. In general, however, the condition is too weak. This is best illustrated by the following example: Consider an operator f defined by the following two rules, where a and c are particular actions in Vis,

$$\frac{X \xrightarrow{a} X'}{f(X,Y) \xrightarrow{a} \mathbf{0}}\ r_a \qquad\qquad \frac{Y \xrightarrow{c} Y'}{f(X,Y) \xrightarrow{c} \mathbf{0}}\ r_c$$

together with their associated τ-rules τ_1 and τ_2, and the following ordering:

$$r_c < r_a,\ \tau_1$$
$$\tau_2 < r_a,\ \tau_1$$
$$r_a < r_c,\ \tau_2$$

We easily check that $<$ satisfies the above condition. Now consider $f(a.\mathbf{0}, \mathbf{0})$ and $f(a.\mathbf{0}, \tau.\mathbf{0})$. We have $f(a.\mathbf{0}, \mathbf{0}) \xrightarrow{a}$ by r_a since r_c and τ_2 are not applicable as $\mathbf{0} \stackrel{\tau}{\not\rightarrow}\stackrel{c}{\not\rightarrow}$. But, $f(a.\mathbf{0}, \tau.\mathbf{0}) \stackrel{a}{\not\rightarrow}$ since $r_a < \tau_2$ and τ_2 is applicable. Also, $f(a.\mathbf{0}, \tau.\mathbf{0}) \stackrel{\tau}{\not\rightarrow}$ since $\tau_2 < r_a$ and r_a is applicable. Hence, although $\mathbf{0}$ and $\tau.\mathbf{0}$ are trace equivalent the process terms $f(a.\mathbf{0}, \mathbf{0})$ and $f(a.\mathbf{0}, \tau.\mathbf{0})$ are not trace equivalent.

Since $\tau_2 < r_a$ we expect, by the above condition, r_a to be above all rules with active second argument. It is also reasonable to expect r_a to be above all rules which are below rules with active second argument. Since $r_a < \tau_2$, this would mean $r_a < r_a$, and thus making $f(a.\mathbf{0}, \mathbf{0})$ and $f(a.\mathbf{0}, \tau.\mathbf{0})$ equivalent. The above condition guarantees this when orderings are transitive, as in [43]. However, we do not assume transitivity here, and so we strengthen the condition as follows:

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

Notice that (5) implies the following limited form of transitivity:

$$\text{if } \ r' < \tau_i < r \ \text{ then } \ r' < r$$

Returning to the operator f above, since $\tau_2 < r_a$ and $2 \in \mathit{active}(\mathit{higher}(r_a))$ we deduce $r_a < r_a$ by (5). Thus, $f(a.\mathbf{0}, \mathbf{0}) \stackrel{a}{\not\rightarrow}$, and $f(a.\mathbf{0}, \mathbf{0})$ and $f(a.\mathbf{0}, \tau.\mathbf{0})$ are trace equivalent.

Unlike our first attempt, condition 5 does not prohibit a τ-rule to be above itself, and thus never enabled. In order to see that this is problematic, consider the the CCS-like renaming operator $[R]$ which renames a by b. The ordering on its rules satisfies $\tau_1 < \tau_1$ and condition (5). Thus, τ_1 is above all action rules for $[R]$. As a result, although $a.\mathbf{0}$ and $\tau.a.\mathbf{0}$ are trace equivalent the processes $(a.\mathbf{0})[R]$ and $(\tau.a.\mathbf{0})[R]$ are not: $(a.\mathbf{0})[R] \xrightarrow{b}$ but $(\tau.a.\mathbf{0})[R] \stackrel{\tau}{\not\rightarrow}$ since τ_1 is never applicable and so $(\tau.a.\mathbf{0})[R] \stackrel{b}{\not\Rightarrow}$. In order to stop this, we impose the following condition that requires that τ-rules are never disabled.

$$\text{not } (\tau_i < \tau_i) \tag{6}$$

There are operators which are definable by ordered bb rules satisfying condition (5) and (6) but which are not well behaved. Consider the priority operator θ (cf [7]) which gives d priority over b. It is defined by the following rule schema r_α, where $\alpha \in \{b, d\}$, with the τ-rule τ_1 and the ordering $r_b < r_d$:

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

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 composition operator. Clearly, p and q are trace equivalent but we have $\theta(p) \stackrel{b}{\Rightarrow}$ and $\theta(q) \stackrel{b}{\not\Rightarrow}$. In order to repair this it is enough to require $r_b < \tau_1$. This example leads to the following condition:

$$\text{if } \; r' < r \text{ and } i \in \mathit{active}(r) \; \text{ then } \; r' < \tau_i \tag{7}$$

The intuition here is that before we apply r' we need to make sure that no other rule with higher priority, and thus their τ-rules, can be applied. In order to see that they are not applicable notice that their active arguments need to be stable.

We claim that conditions (5)–(7) are sufficient to ensure that operators which are defined by bb and eb rules with an ordering satisfying these conditions preserve branching and eager bisimulation preorders respectively. It is possible, however, to obtain slightly more general formats of ordered bb and eb rules that preserve these preorders.

We recall that condition (3) forbids implicit copies in the premises of bb rules, and condition (4) forbids implicit copies in eb rules. We also recall that explicit copies are allowed in both bb and eb rules. However, when bb and eb rules are used with orderings then these conditions can be considerably relaxed. We propose to allow implicit copies in the premises of bb rules and in eb rules provided that these rules are below all their associated τ-rules. In other words, we allow implicit copies of arguments which are stable, i.e. unable to proceed by performing silent actions. The following are the alternatives to (3) and (4):

$$\text{if } \; i \in \mathit{implicit\text{-}copies}(\mathit{pre}(r)) \; \text{ then } \; r < \tau_i \tag{8}$$

$$\text{if } \; i \in \mathit{implicit\text{-}copies}(r) \; \text{ then } \; r < \tau_i \tag{9}$$

Consider the operator $a\text{-}and\text{-}b$ as defined in the previous subsection but with its rules, namely one action rule and one τ-rule, satisfying the above conditions. Then, the process $a\text{-}and\text{-}b(q)$, for any q, can perform c if the τ-rule

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

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

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

$$\text{not } (\tau_i < \tau_i) \tag{6}$$

$$\text{if } \ r' < r \text{ and } i \in \mathit{active}(r) \ \text{ then } \ r' < \tau_i \tag{7}$$

$$\text{if } \ i \in \mathit{implicit\text{-}copies}(\mathit{pre}(r)) \ \text{ then } \ r < \tau_i \tag{8}$$

$$\text{if } \ i \in \mathit{implicit\text{-}copies}(r) \ \text{ then } \ r < \tau_i \tag{9}$$

Fig. 5. Conditions for branching bisimulation ordered and eager bisimulations ordered rules.

for *a-and-b* cannot be applied, and the action rule can be applied. In other words, if $q \overset{\tau}{\not\rightarrow}$, $q \overset{a}{\rightarrow}$ and $q \overset{b}{\rightarrow}$. Hence, for processes p and p' in Figure 1 we have $a\text{-}and\text{-}b(p) \overset{c}{\not\Rightarrow}$, $a\text{-}and\text{-}b(p') \overset{c}{\not\Rightarrow}$ and $a\text{-}and\text{-}b(p) \sqsubseteq_E a\text{-}and\text{-}b(p')$. Similarly, for process q in Figure 3 the τ-rule is always applicable, so $a\text{-}and\text{-}b(q) \overset{c}{\not\Rightarrow}$ as well as $a\text{-}and\text{-}b(p) \overset{c}{\not\Rightarrow}$.

The conditions for branching bisimulation ordered and eager bisimulation ordered rules are gathered for convenience in Figure 5.

Definition 32 A set of OSOS rules (together with the associated orderings) is called bbo, or branching bisimulation ordered, if the rules satisfy conditions (1)–(2) and the ordering satisfies conditions (5)–(8). A set of OSOS rules is called ebo, or eager bisimulation ordered, if the rules satisfy conditions (1)–(2) and the ordering satisfies conditions (5)–(7) and (9). The bbo (ebo) format consists of bbo (ebo) rules and their orderings. An operator is bbo (ebo) if it is defined by bbo (ebo) rules.

This definition, together with Definition 16, gives us the bbo and ebo classes of process languages.

Theorem 33 The bbo class preserves branching bisimulation preorder.

The proof of this theorem is more complicated than the proof of Theorem 30 for the bb format. There are two reasons. Firstly, according to condition (8), bbo rules may have implicit copies of process variables in their premises provided that the relevant arguments are stable. Secondly, bbo rules may have non-trivial orderings associated with them. As a result, when we try to show that property $(B.a)$ of branching bisimulation holds for $f(\boldsymbol{u})$ and $f(\boldsymbol{v})$, we find that we cannot succeed if we assume that *only* properties $(B.a)$, $(B.b)$ and $(B.c)$ of branching bisimulation hold for the related elements of $\boldsymbol{u}$ and $\boldsymbol{v}$. We find that we need to know more about the behaviour of branching bisimilar processes, particularly when one of them is stable.

To illustrate this, consider the following situation: $f(\boldsymbol{u}) \xrightarrow{a} u'$ is derivable by rule r which has $X_l \xrightarrow{a_{l1}} Y_{l1}$ and $X_l \xrightarrow{a_{l2}} Y_{l2}$ among its premises. This means that r has implicit copies of X_l in the premises. Note that bb rules may not have implicit copies. In order to prove $(B.a)$ for $f(\boldsymbol{u})$ and $f(\boldsymbol{v})$ we need to show, among others, $f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} f(\boldsymbol{v}') \xrightarrow{a} v'$ and $u'\mathcal{B}v'$ for some $\boldsymbol{v}'$ and v'. Rule r is applicable to $f(\boldsymbol{u})$, so $u_l \xrightarrow{a_{l1}} u'_{l1}$ and $u_l \xrightarrow{a_{l2}} u'_{l2}$. By $(B.a)$, these transitions imply $v_l \stackrel{\tau}{\Rightarrow} v'_{l1} \xrightarrow{a_{l1}} v''_{l1}$ and $v_l \stackrel{\tau}{\Rightarrow} v'_{l2} \xrightarrow{a_{l2}} v''_{l2}$, and $u_l\mathcal{B}v'_{l1}$ and $u_l\mathcal{B}v'_{l2}$. We are *not* guaranteed that $v'_{l1} = v'_{l2}$. Hence, we are not able to use r with $f(\boldsymbol{v}')$ to derive the required a transition. What we need to do is to deduce $u_l \stackrel{\tau}{\not\rightarrow}$ by (8), and find a property of branching bisimulation that implies the following:

- if $u_l \stackrel{\tau}{\not\rightarrow}$, $u_l \xrightarrow{a_{l1}} u'_{l1}$ and $u_l \xrightarrow{a_{l2}} u'_{l2}$, then $v_l \stackrel{\tau}{\Rightarrow} v'_l \stackrel{\tau}{\not\rightarrow}$, $v'_l \xrightarrow{a_{l1}} v''_{l1}$ and $v'_l \xrightarrow{a_{l2}} v''_{l2}$ for some v'_l and v''_{l1}, v''_{l2}

The property that we seek is $(B.a')$ in Proposition 36 in Appendix A. $(B.a')$ and its two subproperties $(B.a^\star)$ and $(B.c^\star)$ say that, when $p\mathcal{B}q$ and p is stable, any τ-derivative of q (q' such that $q \stackrel{\tau}{\Rightarrow} q'$) is related by $\mathcal{B}$ to p and they have matching visible actions. $(B.c')$ in Proposition 36 is the corresponding property for q when p is convergent, and it will be useful in proving property $(B.c)$. Both $(B.a')$ and $(B.c')$ will be crucial in showing $(B.b)$.

The proof of Theorem 33 proceeds as follows. We use structural induction and show that processes related by $\mathcal{B}$ satisfy properties $(B.a)$–$(B.c')$ of Proposition 36. Since the inductive hypothesis now contains properties that describe the related processes in more detail, we are able to show $(B.b)$ for $f(\boldsymbol{u})$ and $f(\boldsymbol{v})$ within the inductive step. Recall that, in the proof of Theorem 30, we were not able to prove $(B.b)$ this way because we used the less revealing original properties of branching bisimulation. The full details of the proof are given in Appendix A.

Theorem 34 The ebo class preserves eager bisimulation preorder.

The proof follows the same steps as the proof of Theorem 33. Similarly as for branching bisimulation, we need an alternative characterisation of eager bisimulation. The details are given in Appendix B.

Since the ordering on rules is defined implicitly, namely an arbitrary ordering which satisfies conditions (5)–(7), it may be easier to think about ordered rules in terms of their explicit representations. In this representation, called *derived* rules, all rules have the same priority and the 'order of application' information is expressed via negative premises. The form of the derived rules for ebo and bbo rules is informally described as follows:

- The derived action rules corresponding to ebo rules are like ISOS rules, except that they additionally allow *stable* implicit copying, i.e. implicit copies of variables X_i provided there are refusal transitions for X_i as, for example, $X_i \stackrel{\tau}{\not\to}\stackrel{a}{\not\to}$.
- The derived action rules for bbo rules are like ISOS rules but they also allow implicit copies of process variables in the targets and stable implicit copying in the premises.
- The derived τ-rules for both ebo and bbo rules may have, apart from the usual premises, a number of refusal transitions in the premises.

As an illustration, we present below the set of derived rule schemas for the sequential composition operator defined in the Introduction. Here, $\alpha \in \mathsf{Act}$ and $a \in \mathsf{Act} \setminus \{\tau\}$.

$$\frac{X \stackrel{\alpha}{\to} X'}{X;Y \stackrel{\alpha}{\to} X';Y} \qquad \frac{Y \stackrel{\tau}{\to} Y' \quad \{X \stackrel{\alpha}{\not\to}\}_{\alpha\in\mathsf{Act}}}{X;Y \stackrel{\tau}{\to} X;Y'} \qquad \frac{Y \stackrel{a}{\to} Y' \quad \{X \stackrel{\alpha}{\not\to}\}_{\alpha\in\mathsf{Act}}}{X;Y \stackrel{a}{\to} Y'}$$

4.4 Languages for Rooted Eager and Branching Bisimulations

The ebo and bbo formats are very general methods for defining process operators which preserve eager and branching bisimulation preorders respectively. However, there are several popular process operators which do not preserve either of the mentioned preorders. Thus, these operators are not definable within the ebo and bbo formats. The most well-known examples of such operators are the CCS $+$ and the ACP left-merge [10,9] operators. Others include several delay operators from timed process languages, for example the *time-out* operator [24] and other delay operators in [28,29], and the versions of the Kleene star operator [6,17,4]. Moreover, there may be situations where other new operators, which do not preserve branching and eager bisimulation preorders, are useful. Therefore, it may be worthwhile to seek a more general method than the one proposed here, which would provide for the majority of all known operators.

A search for such a method will, most likely, begin by choosing a suitable preorder and then finding a format of rules, as general as practicable, which preserves this preorder. Milner [27] considered observation equivalence (known also as weak bisimulation) for CCS. Since the CCS $+$ does not preserve observation equivalence he proposed a slight restriction of it, called *equality* or *observation congruence* (Definition 2, Section 7.2 in [27]), and showed that all operators of CCS including $+$ preserve this equivalence. Of course, observation congruence coincides with the closure of observation equivalence with respect to all CCS contexts. Also, the ACP left-merge and the mentioned delay operators preserve observation congruence. Observation congruence is sometimes also called *rooted weak bisimulation*, for example in [19,13]. A version of branching bisimulation equivalence, which is defined similarly to observation congruence, is called *rooted branching bisimulation* [19]. It is known that the CCS + and the ACP left-merge and the delay operators preserve rooted branching bisimulation.

Bloom [13] proposed the *RWB cool* and *RBB cool* formats of positive rules which preserve rooted weak bisimulation and rooted branching bisimulation respectively. It is an easy exercise to check that the CCS +, the ACP left-merge and the delay operators are both RWB cool and RBB cool operators. We expect that Bloom's approach can be applied to our preorders and formats in a straightforward way: The rooted versions of our preorders can be easily defined. Then, the rooted ebo and bbo formats can be derived using Bloom's idea of partitioning operators into *wild* operators, like the CCS +, and *tame* operators [13]. We do not present the details of this construction here because we believe that an even more general framework is required. The following two operators motivate the need for such framework.

Consider a version of the Kleene star operator [6,4] defined as follows:

$$\frac{X \stackrel{\alpha}{\to} X'}{a^*X \stackrel{\alpha}{\to} X'} \qquad\qquad a^*X \stackrel{a}{\to} a^*X$$

Also, consider the *interrupt* operator [27] defined as follows:

$$\frac{X \stackrel{\alpha}{\to} X'}{X \wedge Y \stackrel{\alpha}{\to} X' \wedge Y} \qquad\qquad \frac{Y \stackrel{\alpha}{\to} Y'}{X \wedge Y \stackrel{\alpha}{\to} Y'}$$

It is easy to see that these operators are not in the RWB cool and RBB cool formats; they do not preserve the standard weak and branching bisimulations, but they preserve rooted weak and branching bisimulations. Notice that in the first rule schema for a^* and in the second rule schema for '$\wedge$' the silent action of a subprocess causes a structural change of the composite process. For this reason both operators do not preserve weak, eager and branching bisimulation equivalences. For, although b and $\tau.b$ are equivalent neither a^*b and $a^*\tau.b$ nor $a \wedge b$ and $a \wedge \tau.b$ are equivalent. But, clearly, both operators preserve the rooted versions of these equivalences. The Kleene star and the interrupt operators do not fit into Bloom's formats because they violate one of the key conditions: no rule can have, on the right-hand side of its conclusion, an operator which does not preserve the relevant bisimulation.

Fokkink [18] proposed the *RBB safe* format of transition rules which preserves rooted branching bisimulation and which is more general than the RBB cool format. The RBB safe format is a subformat of the *panth* format due to Verhoef [47]. It extends the RBB cool format in several respects:

- It allows open terms as well as variables in the left-hand sides of transitions in the premises.
- Negative premises and predicates are also allowed.
- Transitions with silent actions may appear, under some circumstances, in general rules and in τ-rules.
- The targets of rules may have, under some circumstances, wild operators as well as tame operators.

We note that the Kleene star and interrupt operators discussed above are RBB safe operators.

Recently, the first author jointly with Yuen proposed in [45] a new format called rebo (*rooted eager bisimulation ordered*) which extends the ebo format. rebo operators are defined by OSOS that satisfy certain conditions on rules and rule orderings. These conditions are less restrictive than those for ebo operators. In particular, they require that for each active ith argument of a rebo operator f we have either τ-rule τ_i or the *silent choice* rule below amongst the rules for f.

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

Thus, within the rebo format we have the ability to define operators such as the CCS '+', the Kleene star operator, the interrupt operator and the *delay* operator [24]. [45] also introduces the rbbo (*rooted branching bisimulation ordered*) format, which is a corresponding extension of our bbo format.

5 Comparison with Existing Formats and Classes

Firstly, we compare formats for eager bisimulation and standard weak bisimulation.

- The eb format coincides with the positive ISOS format, written as ISOS^+. Also, the simply WB cool format [13] for the standard weak bisimulation is like the eb format except that it also requires other τ-rules apart from those demanded by condition (1).
- The ebo format extends the eb format with stable implicit copies and, via its orderings, with features equivalent to refusal transitions in the premises of action rules and τ-rules. Moreover, the ebo format extends the ISOS format with stable implicit copies.

Although the ebo and ISOS formats do not allow arbitrary implicit copies, it is argued in [40] that the branching behaviour which can be tested by rules with implicit copies can also be tested by ISOS rules, and thus by ebo rules. The idea is that, instead of using implicit copies of process resources, we first produce the copies by applying rules with explicit copies, and only then we use them. The fully WB cool format [13] allows for rules with implicit copies but only when several kinds of auxiliary rules are present. The effect of the auxiliary rules amounts to what we have informally described above: first making copies of process resources and then using them independently.

Finally, we consider formats for branching bisimulation-like relations.

- The bb format extends the ISOS$^+$ format with implicit copies in the targets of rules. It is very similar to the fully BB cool format [13].
- The bbo format extends the bb format with stable implicit copies in the premises, and, via its orderings, with features equivalent to refusal transitions in premises of action and τ-rules. It extends the fully BB cool format with features equivalent to negative premises.
- The RBB safe format [18] is incomparable with the bbo format. On one hand, informally, the RBB safe format, unlike the bbo format, allows predicates and open terms on the left-hand sides of transitions in the premises. On the other hand, unlike RBB safe rules, bbo rules may have copies of wild arguments (in Fokkink's terminology) in the premises as long as these arguments are stable.

The relationship between classes of process languages, which originate from the formats discussed above except for the RBB safe format, is illustrated in Figure 6. In the diagram the arrows indicate proper subset inclusion, and the labels are explained below. Set inclusions in the diagram are explained as follows:

(1) GSOS = OSOS by Theorem 21 and GSOS $\supseteq$ GSOS$^+$ by Definition 3.
(2) OSOS $\supseteq \mathsf{bbo} \supseteq \mathsf{ebo}$ by Definition 32.
(3) GSOS$^+ \supseteq \mathsf{bb} \supseteq \mathsf{eb}$ by Definition 28.
(4) $\mathsf{bbo} \supseteq \mathsf{bb}$ and $\mathsf{ebo} \supseteq \mathsf{eb}$ by Definitions 7, 28 and 32.
(5) eb = ISOS$^+$ and $\mathsf{ebo} \supseteq$ ISOS and by the remarks above.

The labels in the diagram, namely np, τ, ict and ics, are abbreviations for expressions 'negative premises', 'τ in the the premises', 'implicit copies in the target' and 'implicit copies in stable states' respectively. These expressions indicate why the inclusions in the diagram are proper: each of the expressions names a feature of transition rules (operators) which can be expressed in a class but not in its particular subclass. Let $\mathcal{C}$ and $\mathcal{C}'$ be classes of process languages mentioned in the diagram. Then,

(1) $\mathcal{C}' \xrightarrow{np} \mathcal{C}$ means that there is a language in $\mathcal{C}$ which contains an operator, say f, with a rule with negative premises or ordered rules producing the effect of negative premises such that f cannot be defined in $\mathcal{C}'$. An example of such operator is the sequential composition operator from the Introduction.

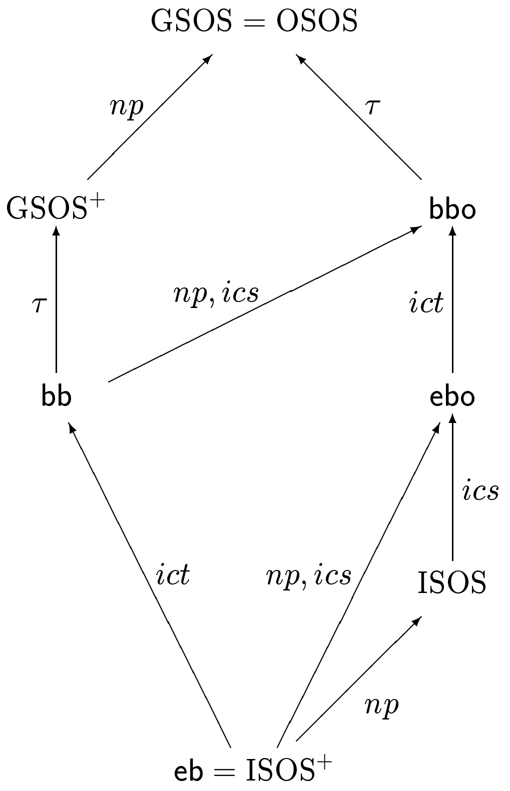

Fig. 6.

(2) $\mathcal{C}' \xrightarrow{\tau} \mathcal{C}$ means that there is a language in $\mathcal{C}$ which contains an operator, say f, with a rule which has τ in the premises and which is not a τ-rule, such that f cannot be defined in $\mathcal{C}'$. An example of such operator is *see-*τ from Section 4.2.

(3) $\mathcal{C}' \xrightarrow{ict} \mathcal{C}$ means that there is a language in $\mathcal{C}$ which contains an operator, say f, with a rule with implicit copies in the target such that f cannot be defined in $\mathcal{C}'$. An example of such operator is *a-then-b* from Section 4.2.

(4) $\mathcal{C}' \xrightarrow{ics} \mathcal{C}$ means that there is a language in $\mathcal{C}$ which contains an operator, say f, with a rule with implicit copying such that f cannot be defined in $\mathcal{C}'$. An example of such operator is a 'stable' version of *a-and-b* which has the extra premise $X \stackrel{\tau}{\not\to}$.

6 Applications

In this section we demonstrate the usefulness of bbo and ebo rules in defining operators which usually require rules with negative premises and copying.

1. We have already seen, in the Introduction, an alternative definition of the sequential composition operator which does not use rules with negative premises.

2. *Action refinement* is an operation which replaces all occurrences of an action by some process. It is known that for purely *sequential processes* action refinement preserves branching bisimulation but not the standard weak bisimulation [19]. We define an ebo action refinement operator *refine-a* such that *refine-a*(p,q) refines all a in p by q. The rules and rule schemas for *refine-a* are given below. We assume $\beta \neq a$ and we use a new visible action $\imath$ to mark the start and the finish of each refinement of a. The reason for using this new visible action rather than, for example, τ will become apparent from the definition of an extended parallel composition operator (with an execution control feature) in the next paragraph.

$$\frac{X \xrightarrow{\beta} X'}{\mathit{refine}\text{-}a(X,Y) \xrightarrow{\beta} \mathit{refine}\text{-}a(X',Y)}\ r_\beta \qquad \frac{X \xrightarrow{a} X'}{\mathit{refine}\text{-}a(X,Y) \xrightarrow{\imath} \mathit{aux}(X',Y,Y)}\ r_a$$

$$\frac{Y \xrightarrow{\alpha} Y'}{\mathit{aux}(X,Y,Z) \xrightarrow{\alpha} \mathit{aux}(X,Y',Z)}\ q_{*\alpha} \qquad \frac{}{\mathit{aux}(X,Y,Z) \xrightarrow{\imath} \mathit{refine}\text{-}a(X,Z)}\ q_\imath$$

The ordering satisfies $q_\imath < q_{*\alpha}$ together with the conditions for ebo rules. $q_\imath < q_{*\alpha}$ implies that rule $q_\imath$ can only be applied when Y is substituted by a deadlocked process, in other words when the refining process has completed its computation. Since *refine-a* is an ebo operator it preserves eager bisimulation. However, it does not preserve the standard weak bisimulation $\approx$. In order to show this consider weak bisimilar processes r and s in Figure 2 and a refinement of action a by the process $a_1a_2.\mathbf{0}$. We have *refine-a*$(s, a_1.a_2.\mathbf{0}) \xrightarrow{\imath} q$ with $q \xrightarrow{a_1}\xrightarrow{a_2}\xrightarrow{\imath}$ *refine-a*$(s', a_1.a_2.\mathbf{0})$ and *refine-a*$(r, a_1.a_2.\mathbf{0}) \xrightarrow{\imath} p$ with $p \xrightarrow{a_1}\xrightarrow{a_2}\xrightarrow{\imath}$ *refine-a*$(r', a_1.a_2.\mathbf{0})$ for some p and q. It is easy to see that $p \not\approx q$ since p may perform b after it performs $a_1a_2\imath$ whereas q will never perform b.

The usefulness of *refine-a* and similarly defined action refinement operators in traditional process languages is limited due to the way they interact with parallel operators. Consider, for example, CCS extended with *refine-a*. We easily check that the processes

$$\mathit{refine}\text{-}a(a.\mathbf{0} \mid b.\mathbf{0},\ a_1.a_2.\mathbf{0}) \quad \text{and} \quad \mathit{refine}\text{-}a(a.b.\mathbf{0} + b.a.\mathbf{0},\ a_1.a_2.\mathbf{0})$$

are strongly bisimilar, where '$|$' is the CCS parallel operator. But, the processes *refine-a*$(a.\mathbf{0} \mid b.\mathbf{0}, a_1.a_2.\mathbf{0})$ and *refine-a*$(a.\mathbf{0}, a_1a_2) \mid b.\mathbf{0}$ are not strongly bisimilar since b may interrupt the refinement of a in the second process. Thus, the second process may perform $\imath a_1 b$, this obviously cannot be performed by the first process. This problem can be overcome by working with process languages which, instead of the CCS parallel operator, use a parallel operator with the *interrupt* function. We do not propose a full solution to this problem but only indicate how it may be achieved. The suggested parallel operator '$\|$' works as '$|$' but it also allows either of its operands to seize and release the control of execution by the special action $\imath$. The set of rules for '$\|$' consists of rules corresponding to those for the CCS parallel operator and the actions different than $\imath$, and of the following rules and rule schemas:

$$\frac{X \xrightarrow{\imath} X'}{X \parallel Y \xrightarrow{\imath} X' \rfloor Y} \qquad \frac{X \xrightarrow{\alpha} X'}{X \rfloor Y \xrightarrow{\alpha} X' \rfloor Y} \qquad \frac{X \xrightarrow{\imath} X'}{X \rfloor Y \xrightarrow{\imath} X' \parallel Y}$$

$$\frac{Y \xrightarrow{\imath} Y'}{X \parallel Y \xrightarrow{\imath} X \lfloor Y'} \qquad \frac{Y \xrightarrow{\alpha} Y'}{X \lfloor Y \xrightarrow{\alpha} X \lfloor Y'} \qquad \frac{Y \xrightarrow{\imath} Y'}{X \lfloor Y \xrightarrow{\imath} X \parallel Y'}$$

We easily verify that *refine-a*$(a \parallel b, a_1 a_2)$ and *refine-a*$(a, a_1 a_2) \parallel b$ are indeed strongly bisimilar.

3. *Priorities* are used in process languages to represent such phenomena as time-outs and interrupts. The first to use priorities were Baeten, Bergstra and Klop [7]. For a given partial order $<$ on visible actions $\theta(X)$ is a restriction of X such that action a can only happen only if no b with $a < b$ is possible. If $B_a = \{b \mid a < b\}$, then this can be defined as follows:

$$\frac{X \xrightarrow{a} X' \quad \{X \stackrel{b}{\nrightarrow}\}_{b \in B_a}}{\theta(X) \xrightarrow{a} \theta(X')}$$

However, θ can be defined by simple ebo rules. Consider the following rule schema, where $c \in \mathsf{Vis}$, and its τ-rule:

$$\frac{X \xrightarrow{c} X'}{\theta(X) \xrightarrow{c} \theta(X')}\ r_c \qquad\qquad \frac{X \xrightarrow{\tau} X'}{\theta(X) \xrightarrow{\tau} \theta(X')}\ \tau_1$$

Let the ordering $<$ be such that $r_a < r_b$ whenever $a < b$, and the ebo conditions are satisfied. Hence, by condition 7 we have $r_a < \tau_1$, for each a, whenever there exists a rule r_b such that $r_b \in \mathit{higher}(r_a)$. Clearly, θ is an ebo operator.

In [32], the second author surveyed a number of approaches to priorities in process algebras. He showed how these approaches can be fitted into a common operational framework based on ordered rules.

4. The next example concerns operators in process languages with discrete time. The *maximal progress* property, as described for example in [49,24], can be expressed as $p \xrightarrow{\tau}$ implies $p \stackrel{\sigma}{\nrightarrow}$, where $X \xrightarrow{\sigma} X'$ denotes the passage of one time unit. The property means that a process blocks the passage of time when it is not stable. Consider a discrete time process language L which satisfies the maximal progress property and does not contain the CCS '$|$'. We extend

$$\frac{T \xrightarrow{\mathsf{s}} T'}{P \mid T \xrightarrow{\mathsf{s}} \mathbf{0}}\ r_{\mathsf{s}}$$

$$\frac{P \xrightarrow{a} P' \quad T \xrightarrow{a} T'}{P \mid T \xrightarrow{\tau} P' \mid T'}\ r_{aa} \qquad \frac{T \xrightarrow{a} T'}{P \mid T \xrightarrow{\tau} \mathbf{0}}\ r_{*a}$$

$$\frac{P \xrightarrow{a} P' \quad T \xrightarrow{a_r} T'}{P \mid T \xrightarrow{\tau} \mathbf{0}}\ r_{aa_r} \qquad \frac{T \xrightarrow{a_r} T'}{P \mid T \xrightarrow{\tau} P \mid T'}\ r_{*a_r}$$

$$\frac{T \xrightarrow{l} T_l \qquad T \xrightarrow{r} T_r}{P \mid T \xrightarrow{\tau} (P \mid T_l) \wedge (P \mid T_r)}\ r_{lr}$$

$$\frac{X \xrightarrow{\mathsf{s}} X'}{X \wedge Y \xrightarrow{\tau} Y}\ q_{\mathsf{s}*} \qquad \frac{Y \xrightarrow{\mathsf{s}} Y'}{X \wedge Y \xrightarrow{\tau} X}\ q_{*\mathsf{s}} \qquad \frac{}{X \wedge Y \xrightarrow{\tau} \mathbf{0}}\ q$$

Fig. 7.

L with '$|$', and denote its synchronisation rule by $r_{a\bar{a}}$. Then, the rule below specifies the passage of time for '$|$' and the ordering $r_\sigma < r_{a\bar{a}}, \tau_1, \tau_2$ guarantees that the maximal progress property holds for the extended language.

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

Clearly, '$|$' is an ebo operator. A general framework for extending ebo process languages with discrete time is proposed in [44]. There, several constraints on ebo rules and their orderings are developed such that timed processes satisfy some of the following properties: time determinacy, maximal progress, patience, weak timelock freeness, timelock freeness and time persistence. A different approach for producing process languages with timed operators is pursued in [33]. It can be shown that the method used for introducing timed operators in [33] can be easily fitted into the presented framework.

5. The last example relates to *refusal simulation preorder* [40,41]. Refusal simulation preorder is preserved by all ISOS operators [41]. Moreover, it coincides, under some mild assumptions, with ISOS trace precongruence [41]. As a result of the discussion in Section 5 concerning the comparison of the ISOS and ebo formats, we have the following:

Proposition 35 All ebo operators preserve refusal simulation preorder. Refusal simulation preorder coincides with ebo trace precongruence.

The proof of the second part of the proposition relies on the fact that, under mild assumptions, refusal simulation preorder coincides with *copy+refusal testing preorder* [40,41]. In the proof we encode copy+refusal tests as ebo processes and internalise testing via executions of ebo contexts—this requires rules with implicit copying.

As this is only an example of the usefulness of ebo rules, we will only consider internalising the *may* copy+refusal tests; the remaining *must* copy+refusal tests can be internalised in a similar way. The *may* tests are generated by the following grammar: $t ::= \mathsf{s} \mid at \mid \tilde{a}t \mid t \wedge t'$. We recall that s is success, $\tilde{a}t$ denotes the test to detect a refusal of a and then to continue testing with t, at is the test for a followed by testing with t, and $t \wedge t'$ is the test which makes two copies of a process, tests them with t and t' respectively and then succeeds if both tests succeeded. Given any ebo language G, we internalise testing by conservatively extending the language with extra ebo operators, which are needed to encode tests as processes and to construct the testing context. The tests are encoded by the function $()^*$ defined as follows:

$$\begin{aligned} \mathsf{s}^* &= \mathsf{s}.\mathbf{0} \\ (at)^* &= a.t^* \\ (\tilde{a}t)^* &= a_r.t^* \\ (t \wedge t')^* &= l.t^* \boxplus r.t'^* \end{aligned}$$

Symbols s, l, r and all a_r are the new prefixing operators and, thus, new action labels and $\boxplus$ is the external choice operator as, for example, in [23]. In order to construct the testing context we need two auxiliary binary operators '$|$' and '$\wedge$' that we define in Figure 7. Here, '$|$' is not, of course, the CCS '$|$'. The associated τ-rules for these operators, which are not shown here, are denoted by $r_{\tau *}, r_{* \tau}$ and $q_{\tau *}, q_{* \tau}$ respectively. In the rules and rule schemas in Figure 7 variable P stands for the tested process and variable T for the test. The ordering $<$ satisfies $r_{*a} < r_{aa}$ and $r_{*a_r} < r_{aa_r}$ and $q < q_{\mathsf{s}*}, q_{*\mathsf{s}}$, plus the usual conditions on ebo rules.

We can deduce $r_{lr} < r_{*\tau}$ by condition (9). It is safe to use r_{lr}, the rule with implicit copying, since the encodings of the tests are τ-free processes, thus stable in all states. Moreover, we obtain $r_{*a} < r_{\tau *}$ by condition (7). Hence, rule r_{*a} can be applied if neither r_{aa} nor $r_{\tau *}$ are applicable, i.e. $P \stackrel{\tau}{\not\rightarrow} \stackrel{a}{\not\rightarrow}$. The testing context is $P \mid T$. We can show that if $p \mid t \stackrel{\mathsf{s}}{\Rightarrow}$, then p passes the test t for some process p and test t.

7 Conclusion

We have introduced a new method for defining arbitrary process operators which normally require rules with negative premises. Our method relies only on rules with positive premises but we equip the rules with an ordering. The ordering specifies the order of application of rules when deriving transitions of processes. We have proposed four formats of rules with silent actions, and thus four classes of process languages, and we have proved that divergence sensitive versions of eager and branching bisimulation preorders are preserved by relevant formats/classes. We have illustrated the usefulness of our method by giving intuitive definitions of sequential composition, action refinement and the copy+refusal testing system.

Acknowledgements

We would like to thank the two referees for their constructive comments. We also thank Shoji Yuen, Frits Vaandrager, Matthew Hennessy and Mike Smyth for their helpful suggestions. Thanks are also due to Paul Taylor for his LaTeX diagram and prooftree macros.

A large part of the research presented here was carried out when the first author held a research post at the Research Institute for Mathematical Sciences of Kyoto University, Japan, at the invitation of Professor Reiji Nakajima.

A A Proof of Congruence Result for bbo Class

The following proposition is vital in the proof of Theorem 33.

Proposition 36 Given the LTS $(\mathrm{T}(\Sigma), \mathsf{Act}, \rightarrow)$, a relation $B \subseteq \mathrm{T}(\Sigma) \times \mathrm{T}(\Sigma)$ is a branching bisimulation if and only if, for all p and q such that pBq, the following properties hold for all appropriate α, a and p', p'', q' and q'':

$$(B.a) \qquad p \xrightarrow{\alpha} p' \text{ implies } \exists q', q''.(q \xRightarrow{\tau} q' \xrightarrow{\widehat{\alpha}} q'' \text{ and } pBq' \text{ and } p'Bq')$$

$$(B.b) \qquad p \stackrel{\tau}{\not\rightarrow} \text{ implies } q\Downarrow$$

$$(B.c) \qquad p\Downarrow \text{ and } q \xrightarrow{\alpha} q' \text{ implies } \exists p', p''.(p \xRightarrow{\tau} p' \xrightarrow{\widehat{\alpha}} p'' \text{ and } p'Bq \text{ and } p''Bq')$$

$$
\begin{array}{llll}
(B.a') & p \stackrel{\tau}{\not\rightarrow} & \text{implies} & [\ q \stackrel{\tau}{\Rightarrow} q' \text{ implies } (pBq' \text{ and} \\
& & (B.a^\star) & p \stackrel{a}{\rightarrow} p' \text{ implies } \exists q'', q'''.(q' \stackrel{\tau}{\Rightarrow} q'' \stackrel{a}{\rightarrow} q''' \\
& & & \qquad\qquad \text{and } p'Bq''') \\
& & (B.c^\star) & q' \stackrel{a}{\rightarrow} q'' \text{ implies } \exists p'.(p \stackrel{a}{\rightarrow} p' \text{ and } p'Bq''))] \\
(B.c') & p\Downarrow \text{ and } q \stackrel{\tau}{\not\rightarrow} & \text{implies} & [\ p \stackrel{\tau}{\Rightarrow} p' \text{ implies } (p'Bq \text{ and} \\
& & (B.a^\dagger) & p' \stackrel{a}{\rightarrow} p'' \text{ implies } \exists q'.(q \stackrel{a}{\rightarrow} q' \text{ and } p''Bq') \\
& & (B.c^\dagger) & q \stackrel{a}{\rightarrow} q' \text{ implies } \exists p'', p'''.(p' \stackrel{\tau}{\Rightarrow} p'' \stackrel{a}{\rightarrow} p''' \\
& & & \qquad\qquad \text{and } p'''Bq'))]
\end{array}
$$

The proposition is proved in the straightforward fashion using the definition of branching bisimulation. Notice, that in $(B.a^\star)$ and $(B.c^\dagger)$ we do not include pBq'' and $p''Bq$ respectively, as might be expected by Definition 25, since these relationships are guaranteed by the first lines of $(B.a')$ and $(B.c')$ respectively.

Proof of Theorem 33 Let $G = (\Sigma, \mathsf{Act}, \rightarrow)$ be any bbo process language. Define a relation $\mathcal{B} \subseteq \mathrm{T}(\Sigma) \times \mathrm{T}(\Sigma)$ as the least relation such that $\boldsymbol{u}\mathcal{B}\boldsymbol{v}$ if $\boldsymbol{u} \sqsubseteq_{\approx_B} \boldsymbol{v}$, and $C[\boldsymbol{u}]\mathcal{B}C[\boldsymbol{v}]$ if $\boldsymbol{u}\mathcal{B}\boldsymbol{v}$, where $C[\boldsymbol{X}]$ is a Σ context and $\boldsymbol{u}$ and $\boldsymbol{v}$ are vectors of closed Σ terms. We need to show that $\mathcal{B}$ is a branching bisimulation relation. The proof is by induction on the structure of process terms. Assume $p\mathcal{B}q$. Instead of showing that p and q satisfy the properties in the definition of branching bisimulation we will show that they satisfy properties $(B.a)$–$(B.c')$ of Proposition 36. If $p \sqsubseteq_{\approx_B} q$, then we are done. Otherwise, as explained at the beginning of the proof of Theorem 30, p and q can be represented as $f(\boldsymbol{u})$ and $f(\boldsymbol{v})$ respectively, where $f \in \Sigma_n$, $\boldsymbol{u}$ and $\boldsymbol{v}$ are of length n and $\boldsymbol{u}\mathcal{B}\boldsymbol{v}$. As the inductive hypothesis, we assume properties $(B.a)$–$(B.c')$ hold for all the corresponding processes in $\boldsymbol{u}$ and $\boldsymbol{v}$ that are related by $\mathcal{B}$. In the following, we prove that the properties of Proposition 36 hold for $f(\boldsymbol{u})$ and $f(\boldsymbol{v})$.

Property $(B.a)$. In fact we show that the following stronger property holds for $f(\boldsymbol{u})$ and $f(\boldsymbol{v})$ and for all μ-derivatives u' of $f(\boldsymbol{u})$.

$$
\begin{array}{ll}
f(\boldsymbol{u}) \stackrel{\mu}{\rightarrow} u' \text{ implies } \exists D[\boldsymbol{X}, \boldsymbol{Y}], \boldsymbol{u}', \boldsymbol{v}', \boldsymbol{v}''. & (f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} f(\boldsymbol{v}') \stackrel{\widehat{\mu}}{\rightarrow} D[\boldsymbol{v}', \boldsymbol{v}''] \\
& \text{and } u' = D[\boldsymbol{u}, \boldsymbol{u}'] \\
& \text{and } \boldsymbol{u}\mathcal{B}\boldsymbol{v}' \text{ and } \boldsymbol{u}'\mathcal{B}\boldsymbol{v}'')
\end{array}
$$

This property together with the definition of $\mathcal{B}$ implies $(B.a)$ for $f(\boldsymbol{u})$ and $f(\boldsymbol{v})$. Assume $f(\boldsymbol{u}) \stackrel{\mu}{\rightarrow} u'$. This means that there is either an action rule or a τ-rule that is enabled such that $f(\boldsymbol{u}) \stackrel{\mu}{\rightarrow} u'$ can be derived by the rule. We only consider the first case since the second case follows similarly. Let $f(\boldsymbol{u}) \xrightarrow{\mu} u'$ be derived by a (positive GSOS) rule r

$$\frac{\{\ X_i \xrightarrow{a_{ij}} Y_{ij}\ \}_{i \in I, j \in J_i}}{f(\boldsymbol{X}) \xrightarrow{\mu} E[\boldsymbol{X}, \boldsymbol{Y}]}$$

using a ground substitution ρ defined by $\rho(\boldsymbol{X}) = \boldsymbol{u}$ and $\rho(Y_{ij}) = u_{ij}$ for all i and j. Thus, $(\rho(f(\boldsymbol{X})), \mu, \rho(E[\boldsymbol{X}, \boldsymbol{Y}])) \in \rightarrow$ or equivalently $f(\boldsymbol{u}) \xrightarrow{\mu} \rho(E[\boldsymbol{X}, \boldsymbol{Y}])$. Let $\boldsymbol{u}'$ denote the vector of all u_{ij}, so $\rho(\boldsymbol{Y}) = \boldsymbol{u}'$. Hence, the required context $D[\boldsymbol{X}, \boldsymbol{Y}]$ is simply $E[\boldsymbol{X}, \boldsymbol{Y}]$ and $u' = E[\boldsymbol{u}, \boldsymbol{u}']$. Moreover, the premises of r are valid under ρ in $\rightarrow$, i.e. $u_i \xrightarrow{a_{ij}} u_{ij}$ for all appropriate i and j.

According to Definition 32, bbo rules may have implicit copies in the premises provided that these rules are below the relevant τ-rules. For the above rule r we assume that there are implicit copies in the premises of arguments X_i, where $i \in K$ for some $\emptyset \neq K \subseteq I$. In other words, $K = \{i \mid \mid J_i \mid > 1\}$ and $\mid J_i \mid = 1$ for each $i \in I \setminus K$. For simplicity we assume $J_i = \{1\}$ for all $i \in I \setminus K$.

Condition (8) requires that $\{\tau_k \mid k \in K\} \subseteq \mathit{higher}(r)$, hence K is a subset of $\mathit{active}(\mathit{higher}(r))$. Since r is enabled at $f(\boldsymbol{u})$ we deduce, by Definition 9, that none of the rules in $\mathit{higher}(r)$ is applicable. Condition (7) tells us that all τ-rules that are associated with the rules in $\mathit{higher}(r)$ are members of $\mathit{higher}(r)$. This implies $\rho(X_k) \stackrel{\tau}{\nrightarrow}$, thus $u_k \stackrel{\tau}{\nrightarrow}$ for all $k \in \mathit{active}(\mathit{higher}(r))$.

Now, we use the inductive hypothesis for the pairs of $\mathcal{B}$ related elements in $\boldsymbol{u}$ and $\boldsymbol{v}$. For each $i \in K$ and $j \in J_i$ we have $u_i \xrightarrow{a_{ij}} u_{ij}$. Also, since $u_i \stackrel{\tau}{\nrightarrow}$ we obtain, by $(B.b)$ and $(B.a')$ of the inductive hypothesis,

$$v_i \stackrel{\tau}{\Rightarrow} v_i' \stackrel{\tau}{\nrightarrow} \quad \text{and} \quad u_i \mathcal{B} v_i'.$$

For all $i \in K$ and relevant j the above transition and $u_i \xrightarrow{a_{ij}} u_{ij}$ imply, by $(B.a^\star)$,

$$v_i \stackrel{\tau}{\Rightarrow} v_i' \xrightarrow{a_{ij}} v_{ij}' \quad \text{and} \quad u_i \mathcal{B} v_i' \quad \text{and} \quad u_{ij} \mathcal{B} v_{ij}'.$$

For each $i \in I \setminus K$ $u_i \xrightarrow{a_{i1}} u_{i1}$ implies, by property $(B.a)$,

$$v_i \stackrel{\tau}{\Rightarrow} v_i' \xrightarrow{a_{i1}} v_{i1}' \quad \text{and} \quad u_i \mathcal{B} v_i' \quad \text{and} \quad u_{i1} \mathcal{B} v_{i1}'.$$

Let $\boldsymbol{v}'$ stand for the sequence $v_1', \ldots, v_n'$ such that $v_k' = v_k$ for $k \notin I \cup \mathit{active}(\mathit{higher}(r))$. Hence, we obtain the required $\boldsymbol{u} \mathcal{B} \boldsymbol{v}'$. Moreover, by letting $\boldsymbol{v}''$ to be the vector of all v_{ij}', where $\boldsymbol{v}''$ is constructed in a corresponding way to $\boldsymbol{u}'$, we obtain $\boldsymbol{u}' \mathcal{B} \boldsymbol{v}''$.

We shall write $\boldsymbol{v} \stackrel{\tau}{\Rightarrow} \boldsymbol{v}^\star$ to mean $v_i \stackrel{\tau}{\Rightarrow} v_i^\star$ for all components v_i of $\boldsymbol{v}$. Hence, $\boldsymbol{v} \stackrel{\tau}{\Rightarrow} \boldsymbol{v}'$. With the notation as in the proof so far, we have the following claim:

Claim 37 If $\boldsymbol{v} \stackrel{\tau}{\Rightarrow} \boldsymbol{v}^\star$ and $r' \in \mathit{higher}(r)$ is not a τ-rule, then r' does not apply to $f(\boldsymbol{v}^\star)$.

Proof. We know that $u_k \stackrel{\tau}{\not\rightarrow}$ for all $k \in \mathit{active}(\mathit{higher}(r))$. Thus, $\boldsymbol{u}\mathcal{B}\boldsymbol{v}$ implies $\boldsymbol{u}\mathcal{B}\boldsymbol{v}^\star$ by $(B.a')$. So, if r' applies to $f(\boldsymbol{v}^\star)$, then r' applies to $f(\boldsymbol{u})$ by $(B.c^\star)$ of the inductive hypothesis. This contradicts the earlier assumption that r is enabled at $f(\boldsymbol{u})$. $\Box$

Now we return to the proof of $(B.a)$. We only need to show $f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} f(\boldsymbol{v}')$ and $f(\boldsymbol{v}') \stackrel{\mu}{\rightarrow} v''$ for appropriate v''.

$f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} f(\boldsymbol{v}')$ is obtained from $\boldsymbol{v} \stackrel{\tau}{\Rightarrow} \boldsymbol{v}'$ by applying τ-rules in $\{\tau_k \mid k \in I \cup \mathit{active}(\mathit{higher}(r))\}$. Our task is to show that at each stage in the derivation of $f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} f(\boldsymbol{v}')$ some of the τ-rules described above are enabled. Suppose that having reached $f(\boldsymbol{v}^\star)$, i.e. $f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} f(\boldsymbol{v}^\star)$, we wish to use one of the transitions $v^\star_m \stackrel{\tau}{\rightarrow} v^{\star\prime}_m$, for $m \in M \subseteq I \cup \mathit{active}(\mathit{higher}(r))$, to derive the next τ transition of $f(\boldsymbol{v}^\star)$. Clearly, all rules τ_m, for $m \in M$, apply to $f(\boldsymbol{v}^\star)$. We use any of them that is enabled. The only problem is if none of these rules is enabled. Suppose for a contradiction that this is the case. Let $m_1 \in M$. Since τ_{m_1} is disabled there must be r' such that $\tau_{m_1} < r'$ and r' applies. But $r < r'$ by (5). So by Claim 37 rule r' is in fact a τ-rule, say τ_{m_2}, so $r < \tau_{m_2}$. Hence, $m_2 \in \mathit{active}(\mathit{higher}(r))$ and since τ_{m_2} applies we obtain $m_2 \in M$. By iterating this procedure we generate a sequence $\tau_{m_1} < \ldots < \tau_{m_n} < \ldots$ with $m_n \in M$. Since M is finite the sequence must contain repeated elements. Namely, the sequence is $\ldots \tau_{m_k} < \ldots < \tau_{m_k} \ldots$. By repeatedly applying the limited form of transitivity (implied by (5)) we obtain $\tau_{m_k} < \tau_{m_k}$. This contradicts (6).

$f(\boldsymbol{v}') \stackrel{\mu}{\rightarrow} v''$ is obtained by rule r with the substitution ρ' defined by $\rho'(\boldsymbol{X}) = \boldsymbol{v}'$ and $\rho'(Y_{ij}) = v'_{ij}$ for all relevant i and j. This is possible because the premises of r are valid under ρ' in $\rightarrow$, namely $v'_i \stackrel{a_{ij}}{\rightarrow} v'_{ij}$ for all i and j. Furthermore, as explained above, none of the rules in $\mathit{higher}(r)$ can be applied to derive transitions of $f(\boldsymbol{v}')$. Finally, we obtain $v'' = E[\boldsymbol{v}', \boldsymbol{v}'']$, where $\boldsymbol{v}''$ denotes the vector of all v'_{ij}. This completes the proof that property $(B.a)$ holds for $f(\boldsymbol{u})$ and $f(\boldsymbol{v})$.

Property $(B.c)$. The proof that $(B.c)$ holds for $f(\boldsymbol{u})$ and $f(\boldsymbol{v})$ is very similar. The important difference is that, given $f(\boldsymbol{u})\Downarrow$, in order to use induction on the related elements of $\boldsymbol{u}$ and $\boldsymbol{v}$ we need to establish that $u_i\Downarrow$ for all i and r such that $i \in \mathit{active}(r)$, $r \in \mathit{rules}(f)$ and r is enabled at $f(\boldsymbol{v})$. We prove this by a contradiction. We assume that $u_i\Uparrow$ for some $i \in \mathit{active}(r)$ where r is enabled at $f(\boldsymbol{v})$. By considering whether or not τ_i is enabled at $f(\boldsymbol{u})$, we deduce, using conditions (5) and (7) and property $(B.a')$, that either $f(\boldsymbol{u})\Uparrow$ or r is not enabled at $f(\boldsymbol{v})$, thus achieving required contradictions.

In the actual proof of $(B.c)$ we use as inductive hypothesis property $(B.c')$, with its subproperties $(B.a^\dagger)$ and $(B.c^\dagger)$, instead of $(B.a')$ and its subproperties. Moreover, we require a claim corresponding to Claim 37 that deals with the τ-derivatives of $\boldsymbol{u}$ instead of the τ-derivatives of $\boldsymbol{v}$. This concludes the case for $(B.c)$.

Property $(B.a')$. We shall need the following result in the proof of $(B.a')$. It will also be vital in the proofs of $(B.b)$ and $(B.c')$.

Claim 38 If $f(\boldsymbol{u})\mathcal{B}f(\boldsymbol{v})$, $f(\boldsymbol{u}) \stackrel{\tau}{\not\to}$ and $f(\boldsymbol{v}) = q_0 \stackrel{\tau}{\to} q_1 \stackrel{\tau}{\to} \cdots \stackrel{\tau}{\to} q_n$, for $n \geq 1$, then $q_i \stackrel{\tau}{\to} q_{i+1}$ is derived by a τ-rule for $0 \leq i < n$.

Proof. Assume $f(\boldsymbol{u})\mathcal{B}f(\boldsymbol{v})$, $f(\boldsymbol{u}) \stackrel{\tau}{\not\to}$ and $f(\boldsymbol{v}) = q_0 \stackrel{\tau}{\to} q_1 \stackrel{\tau}{\to} \cdots \stackrel{\tau}{\to} q_n$ for some $n \geq 1$. The proof is by course of values induction.

Firstly, we argue that $f(\boldsymbol{v}) \stackrel{\tau}{\to} q_1$ is derived by a τ-rule. Assume for a contradiction that an action rule r with the action τ is enabled at $f(\boldsymbol{v})$. The premises of r are valid for $f(\boldsymbol{v})$, meaning $v_k \stackrel{a_{kl}}{\to} v_{kl}$ for all $k \in \mathit{active}(r)$ and appropriate l. Using the following claim, we obtain $u_k \stackrel{\tau}{\not\to}$, and hence $u_k\Downarrow$, for all $k \in \mathit{active}(r)$. Before we continue with this proof, we first show a useful result.

Claim 39 If $f(\boldsymbol{u})\mathcal{B}f(\boldsymbol{v})$, $f(\boldsymbol{u}) \stackrel{\tau}{\not\to}$, $f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} f(\boldsymbol{v}')$ is derivable by τ-rules, and a rule r is enabled at $f(\boldsymbol{v}')$, then $u_k \stackrel{\tau}{\not\to}$ for all $k \in \mathit{active}(r)$.

Proof. Assume for a contradiction that $u_k \stackrel{\tau}{\to}$ for some k. If τ_k is enabled at $f(\boldsymbol{u})$, then $f(\boldsymbol{u}) \stackrel{\tau}{\to}$: contradiction. Otherwise, there exists a rule $r^\dagger$ such that $\mathit{act}(r^\dagger) \neq \tau$, $\tau_k < r^\dagger$ and $r^\dagger$ is enabled at $f(\boldsymbol{u})$. Since $k \in \mathit{active}(r)$, $r < r^\dagger$ by condition (5). Hence, $r < \tau_i$ for $i \in \mathit{active}(r^\dagger)$. This implies $v'_i \stackrel{\tau}{\not\to}$, for all such i, since r is enabled at $f(\boldsymbol{v}')$. $f(\boldsymbol{u}) \stackrel{\tau}{\not\to}$ and $r^\dagger$ is enabled at $f(\boldsymbol{u})$ imply that $u_i \stackrel{\tau}{\not\to}$ for all $i \in \mathit{active}(r^\dagger)$. Moreover, since $f(\boldsymbol{u})\mathcal{B}f(\boldsymbol{v})$, we deduce by the definition of $\mathcal{B}$ that $u_i\mathcal{B}v_i$ for all $i \in \mathit{active}(r^\dagger)$. Now, we shall use properties $(B.a')$ and $(B.a^\star)$ of the theorem's inductive hypothesis. By $(B.a')$, we obtain $u_i\mathcal{B}v'_i$ for all $i \in \mathit{active}(r^\dagger)$ noting that each v'_i is stable as shown above. Since $r^\dagger$ applies to $f(\boldsymbol{u})$, $r^\dagger$ also applies to $f(\boldsymbol{v}')$ by property $(B.a^\star)$ for the pairs of terms u_i and v'_i with $v'_i \stackrel{\tau}{\not\to}$.This contradicts the assumption that r is enabled at $f(\boldsymbol{v}')$ and $r < r^\dagger$. $\square$

We continue with the proof of Claim 38. By $(B.c)$ we obtain $u_k \stackrel{a_{kl}}{\to} u_{kl}$ for all $k \in \mathit{active}(r)$ and appropriate l. This means that r also applies to $f(\boldsymbol{u})$. If r is enabled at $f(\boldsymbol{u})$, then $f(\boldsymbol{u}) \stackrel{\tau}{\to}$ contradicts $f(\boldsymbol{u}) \stackrel{\tau}{\not\to}$. Else, there exists r' such that $r < r'$, $\mathit{act}(r') \neq \tau$ and r' is enabled at $f(\boldsymbol{u})$. This implies $r < \tau_j$ and $v_j \stackrel{\tau}{\not\to}$, for each $j \in \mathit{active}(r')$, since r is enabled at $f(\boldsymbol{v})$. By $(B.a)$, since the premises of r' are valid for $f(\boldsymbol{u})$, they are also valid for $f(\boldsymbol{v})$. This makes r' applicable to $f(\boldsymbol{v})$ which contradicts the assumption that r' is enabled at $f(\boldsymbol{v}')$. This shows that $f(\boldsymbol{v}) \xrightarrow{\tau} q_1$ is by a τ-rule.

Next, we assume that each $q_i \xrightarrow{\tau} q_{i+1}$ is derived by a τ-rule for $0 \leq i < k$, where $k < n$, and prove that $q_k \xrightarrow{\tau} q_{k+1}$ is derived by a τ-rule. Our assumption means that $q_0 \overset{\tau}{\Rightarrow} q_k$ and $q_k = f(\boldsymbol{v}')$ for some $\boldsymbol{v}'$ such that $\boldsymbol{v} \overset{\tau}{\Rightarrow} \boldsymbol{v}'$. In order to show that $f(\boldsymbol{v}') \xrightarrow{\tau} q_{k+1}$ is derived by a τ-rule, we assume for a contradiction, similarly as in the base case, that an action rule r' with $act(r') = \tau$ is enabled at $f(\boldsymbol{v}')$. The premises of r' are valid for $f(\boldsymbol{v}')$: $v'_k \overset{a_{kl}}{\Rightarrow} v'_{kl}$ for all $k \in active(r')$ and appropriate l. By Claim 39, we have $u_k \overset{\tau}{\not\rightarrow}$ for all $k \in active(r')$.

Remark. Notice that although at this point we are able to show $\boldsymbol{u}\mathcal{B}\boldsymbol{v}'$ we cannot use $(B.c)$ and $(B.a)$ to proceed with the proof. The inductive hypothesis guarantees that only those components of $\boldsymbol{u}$ and $\boldsymbol{v}$ that are related by $\mathcal{B}$ satisfy properties $(B.a)$ and $(B.c)$. Their τ-derivatives, which may have grown in size, are not assumed to have these properties. Their behaviour, however, can be deduced from the two more revealing properties $(B.a')$ and $(B.c')$ of our inductive hypothesis.

Returning to the proof of Claim 38, we apply $(B.c^\star)$ to the pairs of terms u_k and v_k. From $v_k \overset{\tau}{\Rightarrow} v'_k$ and $v'_k \overset{a_{kl}}{\Rightarrow} v'_{kl}$ we obtain $u_k \overset{a_{kl}}{\Rightarrow} u_{kl}$ for all $k \in active(r')$ and appropriate l. This implies that r' also applies to $f(\boldsymbol{u})$. If r' is enabled at $f(\boldsymbol{u})$, then $f(\boldsymbol{u}) \xrightarrow{\tau}$ contradicts $f(\boldsymbol{u}) \overset{\tau}{\not\rightarrow}$. Else, there exists r'' such that $r' < r''$, $act(r'') \neq \tau$ and r'' is enabled at $f(\boldsymbol{u})$. This implies $r' < \tau_j$ and $v'_j \overset{\tau}{\not\rightarrow}$ for each $j \in active(r'')$. By $(B.a^\star)$, since the premises of r'' are valid for $f(\boldsymbol{u})$, they are also valid for $f(\boldsymbol{v}')$. This makes r'' applicable to $f(\boldsymbol{v}')$ which contradicts that r' is enabled at $f(\boldsymbol{v}')$. □

Now that Claim 38 is established, we proceed to prove $(B.a')$. Claim 38 tells us that all τ-derivatives of $f(\boldsymbol{v})$ have the form $f(\boldsymbol{v'})$, where $\boldsymbol{v} \overset{\tau}{\Rightarrow} \boldsymbol{v'}$. We show $f(\boldsymbol{u})\mathcal{B}f(\boldsymbol{v}')$ as follows. Assume that in the process of deriving $f(\boldsymbol{v}) \overset{\tau}{\Rightarrow} f(\boldsymbol{v}')$ we only used τ-rules from the set $\{\tau_m \mid m \in M\}$. We easily see, by Claim 39, that $u_m \overset{\tau}{\not\rightarrow}$ for all $m \in M$. This gives us $u_m\mathcal{B}v'_m$, for $m \in M$, by property $(B.a')$. For $i \notin M$ we have $v'_i = v_i$, so $u_i\mathcal{B}v'_i$. Therefore, $\boldsymbol{u}\mathcal{B}\boldsymbol{v}'$, and thus $f(\boldsymbol{u})\mathcal{B}f(\boldsymbol{v}')$ by the definition of $\mathcal{B}$.

The subproperties $(B.a^\star)$ and $(B.c^\star)$ of $(B.a')$ are proved in a very similar fashion as properties $(B.a)$ and $(B.c)$ respectively. In both cases we employ Claim 37 and its corresponding version for the τ-derivatives of $\boldsymbol{u}$.

Property $(B.c')$. Since it is very similar to $(B.a')$ its proof follows closely the proof of $(B.a')$. Naturally, we need versions of Claim 37 and Claim 38 that deal with τ-derivatives of $\boldsymbol{u}$ and $f(\boldsymbol{u})$ respectively. Also, we will be using the $(B.c')$ part of the inductive hypothesis instead of $(B.a')$.

Property $(B.b)$. Suppose for a contradiction that $f(\boldsymbol{v})\Uparrow$. Claim 38 tells us that only τ-rules can be used to produce τ transitions of $f(\boldsymbol{v})$ and of all its τ-derivatives. Let $M \subseteq \mathit{active}(f)$ be the set of τ-rules that can be used to produce any τ transition of any τ-derivative of $f(\boldsymbol{v})$. Since $\mathit{active}(f)$ is finite and $f(\boldsymbol{v})\Uparrow$ we deduce $v_m\Uparrow$ for some $m \in M$. If $u_m \stackrel{\tau}{\not\to}$, then $v_m\Uparrow$ contradicts the $(B.b)$ part of the inductive hypothesis. If $u_m \stackrel{\tau}{\to}$, then since $f(\boldsymbol{u}) \stackrel{\tau}{\not\to}$ the set of rules that are enabled at $f(\boldsymbol{u})$, R, is not empty and τ_m is below a rule r' in R. Let $f(\boldsymbol{v}^\dagger)$ be the first τ-derivative of $f(\boldsymbol{v})$ at which τ_m is enabled. Then, $v_k^\dagger \stackrel{\tau}{\not\to}$ for all $k \in \mathit{active}(r')$. Since r' applies to $f(\boldsymbol{u})$ it also applies to $f(\boldsymbol{v}^\dagger)$ by the fact that $v_k^\dagger \stackrel{\tau}{\not\to}$, for all appropriate k, and by the $(B.a')$ part of the inductive hypothesis. This contradicts the assumption that τ_m is enabled at $f(\boldsymbol{v}^\dagger)$. Therefore, $f(\boldsymbol{v})\Downarrow$ as required. $\Box$

B A Proof of Congruence Result for ebo Class

The following proposition is crucial in the proof of Theorem 34.

Proposition 40 Given the LTS $(\mathrm{T}(\Sigma), \mathsf{Act}, \to)$, a relation $E \subseteq \mathrm{T}(\Sigma) \times \mathrm{T}(\Sigma)$ is an eager bisimulation if and only if, for all p and q such that pEq, the following properties hold for all appropriate actions α, a and processes p', p'', q' and q'':

$$(E.a) \qquad p \stackrel{\alpha}{\to} p' \text{ implies } \exists q', q''.\ (q \stackrel{\tau}{\Rightarrow} q' \stackrel{\widehat{\alpha}}{\to} q'' \text{ and } p'Eq')$$

$$(E.b) \qquad p \stackrel{\tau}{\not\to} \text{ implies } q\Downarrow$$

$$(E.c) \quad p\Downarrow \text{ and } q \stackrel{\alpha}{\to} q' \text{ implies } \exists p', p''.\ (p \stackrel{\tau}{\Rightarrow} p' \stackrel{\widehat{\alpha}}{\to} p'' \text{ and } p''Eq')$$

$$\begin{aligned}
(E.a') \qquad & p \stackrel{\tau}{\not\to} \text{ implies } [q \stackrel{\tau}{\Rightarrow} q' \text{ implies } (pEq' \text{ and} \\
& \quad (E.a^\star) \quad p \stackrel{a}{\to} p' \text{ implies } \exists q'', q'''.(q' \stackrel{\tau}{\Rightarrow} q'' \stackrel{a}{\to} q''' \\
& \qquad\qquad \text{and } p'Eq''') \text{ and} \\
& \quad (E.c^\star) \quad q' \stackrel{a}{\to} q'' \text{ implies } \exists p'.(p \stackrel{a}{\to} p' \text{ and } p'Eq''))] \\
(E.c') \quad & p\Downarrow \text{ and } q \stackrel{\tau}{\not\to} \text{ implies } [p \stackrel{\tau}{\Rightarrow} p' \text{ implies } (p'Eq \text{ and} \\
& \quad (E.a^\dagger) \quad p' \stackrel{a}{\to} p'' \text{ implies } \exists q'.(q \stackrel{a}{\to} q' \text{ and } p''Eq') \text{ and} \\
& \quad (E.c^\dagger) \quad q \stackrel{a}{\to} q' \text{ implies } \exists p'', p'''.(p' \stackrel{\tau}{\Rightarrow} p'' \stackrel{a}{\to} p''' \\
& \qquad\qquad \text{and } p'''Eq')))]
\end{aligned}$$

Proof of Theorem 34 Let $G = (\Sigma, \mathsf{Act}, \rightarrow)$ be any ebo process language. Define a relation $\mathcal{E} \subseteq \mathrm{T}(\Sigma) \times \mathrm{T}(\Sigma)$ as the least relation such that $\boldsymbol{u}\mathcal{E}\boldsymbol{v}$ if $\boldsymbol{u} \sqsubseteq_{\approx_E} \boldsymbol{v}$, and $C[\boldsymbol{u}]\mathcal{E}C[\boldsymbol{v}]$ if $\boldsymbol{u}\mathcal{E}\boldsymbol{v}$, where $C[\boldsymbol{X}]$ is a Σ context and $\boldsymbol{u}$ and $\boldsymbol{v}$ are vectors of closed Σ terms. We need to show that $\mathcal{E}$ is an eager bisimulation relation. By Proposition 40 is enough to show that for all process terms that are related by $\mathcal{E}$ they satisfy properties $(E.a)$–$(E.c')$.

As before the proof is by induction on the structure of process terms. Let $p\mathcal{E}q$ and assume that $(E.a)$–$(E.c')$ hold for all subterms of p and q that are related by $\mathcal{E}$. If $p \sqsubseteq_{\approx_E} q$, then we are done. Else, p and q can be represented as $f(\boldsymbol{u})$ and $f(\boldsymbol{v})$ respectively for some n-ary f, $\boldsymbol{u}$ and $\boldsymbol{v}$ with $\boldsymbol{u}\mathcal{E}\boldsymbol{v}$.

Firstly, we proceed to show that property $(E.a)$ holds for $f(\boldsymbol{u})$ and $f(\boldsymbol{v})$. Similarly as before we show the following stronger property and for all μ-derivatives u' of $f(\boldsymbol{u})$:

$$\begin{aligned} f(\boldsymbol{u}) \xrightarrow{\mu} u' \ \text{ implies } \ \exists D[\boldsymbol{U},\boldsymbol{V}], \boldsymbol{u}^\dagger, \boldsymbol{u}', \boldsymbol{v}', \boldsymbol{v}^\dagger, \boldsymbol{v}''.\ & (f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} f(\boldsymbol{v}') \xrightarrow{\widehat{\mu}} D[\boldsymbol{v}^\dagger, \boldsymbol{v}''] \\ & \text{and } u' = D[\boldsymbol{u}\dagger, \boldsymbol{u}'] \\ & \text{and } \ \boldsymbol{u}^\dagger \mathcal{E} \boldsymbol{v}^\dagger \text{ and } \boldsymbol{u}' \mathcal{E} \boldsymbol{v}'') \end{aligned}$$

Assume $f(\boldsymbol{u}) \xrightarrow{\mu} u'$. This means that there is either an action rule or a τ-rule that is enabled such that $f(\boldsymbol{u}) \xrightarrow{\mu} u'$ can be derived by the rule. We only consider the first case as the second case follows similarly. Let $f(\boldsymbol{u}) \xrightarrow{\mu} u'$ be derived by a rule r

$$\frac{\{\ X_i \xrightarrow{a_{ij}} Y_{ij}\ \}_{i \in I, j \in J_i}}{f(\boldsymbol{X}) \xrightarrow{\mu} E[\boldsymbol{X},\boldsymbol{Y}]}$$

using a ground substitution ρ defined by $\rho(\boldsymbol{X}) = \boldsymbol{u}$ and $\rho(Y_{ij}) = u_{ij}$ for all i and j. Thus, $f(\boldsymbol{u}) \xrightarrow{\mu} \rho(E[\boldsymbol{X},\boldsymbol{Y}])$. Let $\boldsymbol{u}'$ denote the vector of all u_{ij}, so $\rho(\boldsymbol{Y}) = \boldsymbol{u}'$. Also, the premises of r are valid under ρ in $\rightarrow$, namely $u_i \xrightarrow{a_{ij}} u_{ij}$ for all appropriate i and j.

According to Definition 32, ebo rules may have implicit copies in the premises and in the target provided that these rules are below the relevant τ-rules. For the above rule r we assume that there are implicit copies in the premises of arguments X_i, where $i \in K$ for some $\emptyset \neq K \subseteq I$. In other words, $K = \{i \mid\ \mid J_i \mid > 1\}$ and $\mid J_i \mid = 1$ for each $i \in I \setminus K$. For simplicity we assume $J_i = \{1\}$ for all $i \in I \setminus K$. We also assume that there are implicit copies in the target of arguments X_i, where $i \in L$ for some $\emptyset \neq L \subseteq \{1, \ldots, n\}$. Hence, only X_i where $i \in L$ or $i \notin I$ appear in $E[\boldsymbol{X},\boldsymbol{Y}]$. Let $\boldsymbol{X}^\dagger$ be the sequence of all such X_i, and let $\boldsymbol{u}^\dagger = \rho(\boldsymbol{X}^\dagger)$. Moreover, the required context $D[\boldsymbol{U},\boldsymbol{V}]$ is simply $E[\boldsymbol{X}^\dagger,\boldsymbol{Y}](= E[\boldsymbol{X},\boldsymbol{Y}])$ and $u' = E[\boldsymbol{u}^\dagger, \boldsymbol{u}']$.

(8) requires that $\{\tau_k \mid k \in K\} \cup \{\tau_l \mid l \in L\} \subseteq \mathit{higher}(r)$, and hence $K \cup L$ is a subset of $\mathit{active}(\mathit{higher}(r))$. As before we deduce, by Definition 9 and (7),

that $u_k \stackrel{\tau}{\not\rightarrow}$ for all $k \in \mathit{active}(\mathit{higher}(r))$.

Now, we can use the inductive hypothesis for the pairs of $\mathcal{E}$ related elements in $\boldsymbol{u}$ and $\boldsymbol{v}$. For all $i \in \mathit{active}(\mathit{higher}(r))$ $u_i \stackrel{\tau}{\not\rightarrow}$ implies

$$v_i \stackrel{\tau}{\Rightarrow} v_i' \stackrel{\tau}{\not\rightarrow} \quad \text{and} \quad u_i \mathcal{E} v_i' \tag{B.1}$$

by $(E.b)$ and $(E.a')$ of the inductive hypothesis. For each $i \in K$ and $j \in J_i$ transition $u_i \stackrel{a_{ij}}{\rightarrow} u_{ij}$ implies, by (B.1) and $(E.a^\star)$ of the inductive hypothesis,

$$v_i \stackrel{\tau}{\Rightarrow} v_i' \stackrel{a_{ij}}{\rightarrow} v_{ij}' \quad \text{and} \quad u_{ij} \mathcal{E} v_{ij}'.$$

For each $i \in I \setminus K$ $u_i \stackrel{a_{i1}}{\rightarrow} u_{i1}$ implies, by property $(E.a)$ again,

$$v_i \stackrel{\tau}{\Rightarrow} v_i' \stackrel{a_{i1}}{\rightarrow} v_{i1}' \quad \text{and} \quad \text{and} \quad u_{i1} \mathcal{E} v_{i1}'.$$

Let $\boldsymbol{v}'$ stand for the sequence $v_1', \ldots, v_n'$ such that $v_k' = v_k$ when $k \notin I \cup \mathit{active}(\mathit{higher}(r))$. Let $\boldsymbol{v}^\dagger$ stand for the sequence of only those v_i' where $i \in L$ or $i \notin I$ (ordered in the corresponding way to $\boldsymbol{u}^\dagger$). As for the corresponding elements of $\boldsymbol{u}^\dagger$ and $\boldsymbol{v}^\dagger$, we have so far the following:

$$\begin{array}{lll}
(u_i =)\ u_i^\dagger \mathcal{E} v_i^\dagger\ (= v_i) & & \text{for} \quad i \notin I \wedge i \notin \mathit{active}(\mathit{higher}(r)) \\
(u_i =)\ u_i^\dagger \mathcal{E} v_i^\dagger\ (= v_i') & \text{by (B.1)} & \text{for} \quad i \notin I \wedge i \in \mathit{active}(\mathit{higher}(r)) \\
(u_i =)\ u_i^\dagger \mathcal{E} v_i^\dagger\ (= v_i') & \text{by (B.1)} & \text{for} \quad i \in L
\end{array}$$

Therefore, we have $\boldsymbol{u}^\dagger \mathcal{E} \boldsymbol{v}^\dagger$. Moreover, by letting $\boldsymbol{v}''$ to be the vector of all v_{ij}', where $\boldsymbol{v}''$ is constructed in a corresponding way as $\boldsymbol{u}'$, we obtain $\boldsymbol{u}' \mathcal{E} \boldsymbol{v}''$.

Finally, we only need to show $f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} f(\boldsymbol{v}')$ and $f(\boldsymbol{v}') \stackrel{\mu}{\rightarrow} v''$ for some v''. This is done precisely in the same way as in the proof of Theorem 33.

The proofs of properties $(E.b)$–$(E.c')$ hold for $f(\boldsymbol{u})$ and $f(\boldsymbol{v})$ are very similar to those of $(B.b)$–$(B.c')$ for Theorem 30. □

References

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

[2] L. Aceto. Deriving complete inference systems for a class of GSOS languages generating regular behaviours. In *Proceedings of the 5th International Conference CONCUR'94*, volume 789 of *LNCS*. Springer, 1994.

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

[4] L. Aceto, W. Fokkink, R. van Glabbeek, and A. Ingólfsdóttir. Axiomatizing prefix iteration with silent steps. *Information and Computation*, 127:1–52, 1996.

[5] L. Aceto and A. Ingólfsdóttir. CPO models for compact GSOS languages. *Information and Computation*, 129(2):107–141, 1996.

[6] L. Aceto and A. Ingólfsdóttir. An equational axiomatisation of observation congruence for prefix iteration. In M. Wirsing and M. Nivat, editors, *Proceedings of the 5th International Conference on Algebraic Methodology and Software Technology AMAST'96*, volume 1101 of *LNCS*. Springer, 1996.

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

[8] 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.

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

[10] J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. *Information and Control*, 60:109–137, 1984.

[11] B. Bloom. *Ready simulation, bisimulation, and the semantics of CCS-like languages*. PhD thesis, Massachusetts Institute of Technology, 1990.

[12] B. Bloom. Strong process equivalence in the presence of hidden moves. Preliminary report, Massachusetts Institute of Technology, 1990.

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

[14] B. Bloom, S. Istrail, and A.R. Meyer. Bisimulation can't be traced: preliminary report. In *Conference Record of the 15th ACM Symposium on Principles of Programming Languages*, 1988.

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

[16] F.S. de Boer and C. Palamidessi. Embedding as a tool for language comparison. *Information and Computation*, 108(1):128–157, 1994.

[17] W. Fokkink. On the completeness of the equations for the Kleene star in bisimulation. In M. Wirsing and M. Nivat, editors, *Proceedings of the 5th International Conference on Algebraic Methodology and Software Technology AMAST'96*, volume 1101 of *LNCS*. Springer, 1996.

[18] W.J. Fokkink. Rooted branching bisimulation as a congruence. *Journal of Computer and System Sciences*, 60(1):13–37, 2000.

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

[20] R.J. van Glabbeek and W.P. Weijland. Branching time and abstraction in bisimulation semantics. *Journal of the ACM*, 43(3):555–600, 1996.

[21] J.F. Groote. Transition system specifications with negative premises. *Theoretical Computer Science*, 118(2):263–299, 1993.

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

[23] M. Hennessy. *An Algebraic Theory of Processes.* The MIT Press, 1988.

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

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

[26] R. Milner. A modal characterisation of observable machine behaviours. In G. Astesiano and C. Böhm, editors, *Proceedings of CAAP'81*, volume 112 of *LNCS*. Springer, 1981.

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

[28] X. Nicollin and J. Sifakis. An overview and synthesis on timed process algebras. In K.G. Larsen and A. Skou, editors, *Proceedings of CAV'91*, volume 575 of *LNCS*. Springer, 1991.

[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*, Berlin, 1981. Springer.

[31] I.C.C. Phillips. Refusal testing. *Theoretical Computer Science*, 50:241–284, 1987.

[32] I.C.C. Phillips. Approaches to priority in process algebras. Unpublished manuscript, 1994.

[33] I.C.C. Phillips and I. Ulidowski. Stable and timed formats for process algebra. Technical Report, Imperial College, 1995.

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

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

[36] E. Shapiro. Separating concurrenct languages with categories of language embeddings. In *Proceedings of 23rd Annual ACM Symposium on Theory of Computing*, pages 198–208, 1991.

[37] E. Shapiro. Embeddings among concurrent programming languages. In W.R. Cleaveland, editor, *Proceedings of 3rd International Conference CONCUR'92*, volume 630 of *LNCS*, pages 486–503, 1992.

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

[39] D. Turi and G.D. Plotkin. Towards a mathematical operational semantics. In *Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science*, pages 280–291. IEEE, Computer Society Press, 1997.

[40] I. Ulidowski. Equivalences on observable processes. In *Proceedings of the 7th Annual IEEE Symposium on Logic in Computer Science*, pages 148–159. IEEE, Computer Science Press, 1992.

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

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

[43] I. Ulidowski and I.C.C. Phillips. Formats of ordered SOS rules with silent actions. In M. Bidoit and M. Dauchet, editors, *Proceedings of the 7th International Conference on Theory and Practice of Software Development TAPSOFT'97*, volume 1214 of *LNCS*. Springer, 1997.

[44] I. Ulidowski and S. Yuen. Extending process languages with time. In M. Johnson, editor, *Proceedings of the 6th International Conference on Algebraic Methodology and Software Technology AMAST'97*, volume 1349 of *LNCS*. Springer, 1997.

[45] I. Ulidowski and S. Yuen. Process languages for rooted eager bisimulation. In D. Miller and C. Palamidessi, editors, *Proceedings of the 11th International Conference on Concurrency Theory CONCUR2000*, volume 1877 of *LNCS*. Springer, 2000.

[46] 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*. IEEE, Computer Society Press, 1991.

[47] C. Verhoef. A congruence theorem for structured operational semantics with predicates and negative premises. *Nordic Journal of Computing*, 2(2):274–302, 1995.

[48] D. Walker. Bisimulation and divergence. *Information and Computation*, 85(2):202–241, 1990.

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

[50] W.P. Weijland. *Synchrony and Asynchrony in Process Algebra.* PhD thesis, University of Amsterdam, 1989.