Available online at www.sciencedirect.com
SCIENCE DIRECT®
The Journal of Logic and Algebraic Programming 60–61 (2004) 401–460
THE JOURNAL OF LOGIC AND ALGEBRAIC PROGRAMMING
www.elsevier.com/locate/jlap

Process languages with discrete relative time based On the Ordered SOS format and rooted eager bisimulation

Irek Ulidowski[a,*], Shoji Yuen[b]

[a] *Department of Mathematics and Computer Science, University of Leicester, University Road, Leicester LE1 7RH, UK*
[b] *Department of Information Engineering, Graduate School of Information Science, Nagoya University. PRESTO, Japan Science Technology Agency, Nagoya 464-8603, Japan*

Abstract

We propose a uniform framework, based on the Ordered Structural Operational Semantics (OSOS) approach of Ulidowski and Phillips [Information and Computation 178 (1) (2002) 180], for process languages with discrete relative time. Our framework allows the user to select favourite process operators, whether they are standard operators or new application-specific operators, provided that they are OSOS definable and their OSOS rules satisfy several simple conditions. The obtained timed process languages preserve a timed version of rooted eager bisimulation preorder and the time determinacy property. We also propose several additional conditions on the type of OSOS definitions for the operators so that several other properties which reflect the nature of time passage, such as the maximal progress, patience and time persistence properties, are also satisfied.

Keywords: Discrete relative time; Process languages; Structured operational semantics; Formats of SOS rules: Ordered SOS format and rooted eager bisimulation ordered format; Process preorders: rooted eager bisimulation and timed rooted eager bisimulation; Timed process languages; Timed properties: time determinacy, time synchrony, maximal progress, patience, urgency, weak timelock freeness, constancy of offers and time persistence; Congruence results

1. Introduction

Process languages, for example CCS [33], CSP [28] and ACP [11], are well-known formalisms for specification and reasoning about concurrent systems. They are particularly successful in describing the functional behaviour of systems, represented by *actions*,

* Corresponding author. Tel.: +44-116-252-3801; fax: +44-116-252-3604.
E-mail addresses: iu3@mcs.le.ac.uk (I. Ulidowski), yuen@is.nagoya-u.ac.jp (S. Yuen).
URLs: http://www.mcs.le.ac.uk/∼iulidowski/ (I. Ulidowski), http://www.agusa.i.is.nagoya-u.ac.jp/person/yuen (S. Yuen).

1567-8326/$ - see front matter Published by Elsevier Inc.
doi:10.1016/j.jlap.2004.03.002

but they lack the ability to express the *temporal* aspect of the behaviour. This inadequacy has been addressed and a large number of extensions of process languages with *time*, or simply *timed* process languages, have been proposed. These can be broadly divided into process languages with *discrete* time, for example [7,26,27,29,34,37,39], and process languages with *real* (dense) time, for example [6,29–31,42,44,59]. Also, there are process languages that allow a user-specified time domain such as timed μCRL [23,24] and Enhanced Timed-LOTOS (ET-LOTOS) [30,31].

We are convinced that in order to increase the usefulness of process languages in software design and analysis we need to develop a theoretically sound framework for extending process languages with various user-friendly and application-specific features. ET-LOTOS is an excellent example of a successful specification formalism which is based on a process language extended with time as well as with data types, and timed μCRL is another good example. Our ultimate aim is to develop a framework, as general and practicable as possible, for extending traditional process languages, like CCS and CSP, with user-defined and application-specific operators and with the notion of discrete time. The framework is to be based on a general format of SOS rules for the chosen weak semantics. The analysis of the SOS definitions of the existing process languages with time suggests that the format should support features such as *negative information* and *predicates* in SOS rules and, to small extend, *complex terms* in the premises of SOS rules, with negative information being the most crucial and commonly used. However, at present there is no single format for weak semantics that possesses all these features: the subformats of the *GSOS* format of Bloom [13,15] do not permit any of these features, the format of Fokkink [18] based on the *panth* format [56] allows only a restricted use of negative information, and the formats in [47,48,52,53,55] allow negative information but do not include the other two features. As a result this paper presents the outcome of the initial phase of research, where we consider the formats that support negative information most fully, namely the subformats of OSOS rules for the divergence sensitive version of weak bisimulation [52,53,55]. In future it is our intention to extend the OSOS format with predicates and other features, to find subformats for major weak semantics, and to generalise the present work on process languages with discrete relative time.

The ability to use negative information in SOS rules is important because there are many standard process operators which are best defined by such rules, for example sequential composition and priority operators (see Example 8) and action refinement operator. More importantly, rules with negative information are used widely in the majority of process languages with time, either in the definitions of new *timed* operators or in the additional *timed* rules for the standard operators. Such rules are essential in expressing important *timed properties* of processes. For example the maximal progress property in Fig. 2 requires that if a process passes time, then it is stable (cannot perform any τ actions). In Hennessy and Regan's TPL [27] the parallel composition operator of CCS is extended with the following single timed rule (see Example 17), where σ a timed action representing the passage of one time unit.

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

According to the rule, process $p \parallel q$ can pass time if both p and q can pass time and if p and q cannot communicate and cannot evolve silently. The last condition is conveniently represented by the negative premise $X \parallel Y \xrightarrow{\tau}$.

Instead of rules with negative premises, we will use equally expressive *Ordered* SOS rules [53], an extension of the original SOS rules of Plotkin [41], as our method for defining process languages. This choice is motivated carefully in Section 2.4. We feel that it is easier to express how silent and visible actions, negative information and *copying* can be combined safely in transition rules using the OSOS approach (via global conditions for which we have independent motivation) rather than using the traditional GSOS approach.

We shall consider a divergence sensitive version of weak bisimulation relation, called *eager bisimulation* [1,19,32,48,52,53,55,57], as our preorder on processes. For reasons of compositionality and modularity, we insist that all the considered operators preserve this preorder. Further motivation is given in Sections 2.1 and 2.4.

After [27,39], we assume that timed concurrent systems function as collections of components which can either perform actions or let time pass. Actions of the components are instantaneous and they can be either synchronous or independent. The passage of time, which shall be represented by *timed* actions, is synchronous in all active components that are able to pass time. We will also consider several properties of processes which reflect the nature of time passage. These properties are henceforth called the *timed properties*.

Timed process languages which can be defined within our framework have two important features. Firstly, they preserve a timed version of *rooted* eager bisimulation. Secondly, they satisfy the *time determinacy* property. Informally, a timed process language has this property if no process can evolve to two different processes by performing a timed action. We also investigate other timed properties, which we formally define in Section 4, including *urgency, weak timelock freeness, maximal progress, patience* and *time persistence,* and discuss circumstances under which timed process languages satisfy these properties.

Our work provides a flexible specification formalism for timed concurrent systems. It subsumes our preliminary paper [54]. Given a system to specify, the user is free to choose a timed process language that is suitable for the description of the system, by selecting both standard and new application-specific process operators. Compositionality of specifications is guaranteed by our congruence results. Moreover, the axiomatisation algorithms for GSOS and de Simone process languages [2,49,50] can be used, in conjunction with other techniques, to generate sound and complete axiom systems for the chosen timed process language. This may assist the verification stage of system's development.

1.1. Outline

In Section 2 we recall the definitions of labelled transition system and the eager bisimulation relation. We also define the rooted eager bisimulation relation. Also in Section 2, we recall the definitions of the OSOS and GSOS formats and process languages, and compare them. Moreover, we define a very general class of process languages, called rebo process languages, which preserve rooted eager bisimulation. Section 3 presents a uniform method for defining process languages with discrete relative time. We also show that the defined process languages satisfy the time determinacy property and preserve a timed version of rooted eager bisimulation. In Section 4 we discuss several other timed properties and establish conditions under which these properties are satisfied. In Section 5 we examine several existing timed process languages to see how they fit into our framework. The last section contains conclusions and discusses directions for future research.

2. Preliminaries

In this section we recall several basic notions which together constitute the foundation for the results in this paper. As semantics we choose divergence sensitive, timed and rooted version of eager bisimulation. We work with SOS rules in the rebo format [53,55], a subformat of the Ordered SOS [53] format which preserves rooted eager bisimulation. Both of these formats are motivated, defined and illustrated by many examples.

Following the results in [53,55], we could have equally considered formats and timed process languages for rooted branching bisimulation preorder. However, as this work is intended to subsume [54] we continue here with eager bisimulation.

2.1. Rooted eager bisimulation

We model timed concurrent systems by means of *processes* which are states in a *labelled transition system*:

Definition 1. A labelled transition system (or LTS, for short) is a structure $(\mathcal{P}, A, \rightarrow)$, where $\mathcal{P}$ is the set of processes, A is the set of actions and $\rightarrow \subseteq \mathcal{P} \times A \times \mathcal{P}$ is a *transition relation*.

The behaviour of such systems is represented by transitions between processes, where the functional behaviour is modelled by transitions labelled with standard visible or silent actions and the temporal behaviour, or the passage of time, is represented by transitions labelled with a special *timed* action.

The next two paragraphs introduce some standard notation. $\mathcal{P}$ is the set of processes and it is ranged over by $p, q, r, s, \ldots$. Vis is a finite set of visible actions and it is ranged over by a, b, c. Symbol τ denotes the silent action and σ is the timed action which represents the passage of one time unit; $\tau, \sigma \notin \mathsf{Vis}$. $\mathsf{Act} = \mathsf{Vis} \cup \{\tau\}$ is ranged over by α, β and $\mathsf{Act}_\sigma = \mathsf{Act} \cup \{\sigma\}$ is ranged over by χ. No distinction is made between σ and visible actions in this section. Only in Section 3 and afterwards action σ plays its appointed rôle.

We will use the following abbreviations. We write $p \xrightarrow{\chi} q$ for $(p, \chi, q) \in \rightarrow$ and read it as "process p performs χ and in doing so becomes process q". Expressions of the form $p \xrightarrow{\chi} q$ will be called *transitions*. We write $p \xrightarrow{\chi}$ when there is q such that $p \xrightarrow{\chi} q$, and $p \xrightarrow{\chi} \!\!\!\!\!\rightarrow$ otherwise. Expressions $p \stackrel{\tau}{\Rightarrow} q$ and $p \stackrel{\chi}{\Rightarrow} q$, where $\chi \neq \tau$, denote $p(\xrightarrow{\tau})^* q$ and $p(\xrightarrow{\tau})^* p' \xrightarrow{\chi} q$, for some p', respectively. Here, $(\xrightarrow{\tau})^*$ is the transitive reflexive closure of $\xrightarrow{\tau}$. Note that for technical reasons that we explain shortly our notation $\stackrel{\chi}{\Rightarrow}$ has a different meaning from that originally given by Milner in, for example, [33] where it denotes $(\xrightarrow{\tau})^* \xrightarrow{\chi} (\xrightarrow{\tau})^*$. Given a sequence of actions $s = \chi_1 \ldots \chi_n$, where $\chi_i \in \mathsf{Act}_\sigma$ and $1 \leqslant i \leqslant n$, we say that q is an s-derivative of p if $p \stackrel{\chi_1}{\Rightarrow} \cdots \stackrel{\chi_n}{\Rightarrow} q$. The expression $p\Uparrow$, read as "p is divergent", means $p(\xrightarrow{\tau})^\omega$. We say p is convergent, written as $p\Downarrow$, if p is not divergent. We assume that, if $\chi = \tau$ then $p \xrightarrow{\widehat{\chi}} p'$ means $p \xrightarrow{\tau} p'$ or $p \equiv p'$, else it is simply $p \xrightarrow{\chi} p'$.

Definition 2. Given $(\mathcal{P}, \mathsf{Act}_\sigma, \rightarrow)$, a relation $R \subseteq \mathcal{P} \times \mathcal{P}$ is an *eager bisimulation* if, for all $(p, q) \in R$, the following properties hold.

$(E.a) \quad \forall \chi.\ (p \xrightarrow{\chi} p' \text{ implies } \exists q', q''.\ (q \stackrel{\tau}{\Rightarrow} q' \xrightarrow{\widehat{\chi}} q'' \text{ and } p' R q''))$

$(E.b)$ $p\Downarrow$ implies $q\Downarrow$

$(E.c)$ $p\Downarrow$ implies $\forall\chi.\ (q \xrightarrow{\chi} q'$ implies $\exists p', p''.\ (p \stackrel{\tau}{\Rightarrow} p' \xrightarrow{\widehat{\chi}} p''$ and $p''Rq'))$.

We write $p \sqsubseteq_{\approx} q$ if there exists an eager bisimulation R such that pRq.

Example 3. Consider processes p and q defined as follows: $p \xrightarrow{a} \mathbf{0}$, $q \xrightarrow{a} \mathbf{0}$ and $q \xrightarrow{\tau} q$. Process p can perform action a and evolve to the deadlocked process $\mathbf{0}$. Process q can perform a after any number of silent actions, but it can also compute internally by performing silent actions forever. Thus, $q \sqsubseteq_{\approx} p$ but $p \not\sqsubseteq_{\approx} q$. Moreover, consider CCS-like processes $r \equiv a.(b + \tau.c)$ and $s \equiv a.(b + \tau.c) + a.c$. Processes r and s are equivalent according to the weak bisimulation relation of Milner [33] ($s = r$ is an instance of the third τ-law), but $r \not\sqsubseteq_{\approx} s$ since after $s \xrightarrow{a} c$ there are no r' and r'' such that $r \stackrel{\tau}{\Rightarrow} r' \xrightarrow{a} r''$ and $r'' \sqsubseteq_{\approx} c$.

It is clear that $\sqsubseteq_{\approx}$ is a preorder. $\sqsubseteq_{\approx}$ is the version of weak bisimulation studied in [1,19,32,48,52,53,57], where testing, modal logic and axiomatic characterisations were proposed and congruence results with respect to the ISOS, eb and ebo formats were proved. $\sqsubseteq_{\approx}$ coincides with *delay* bisimulation [20,60] for processes with no divergence. We have chosen this finer version of weak bisimulation in preference to the standard weak bisimulation [33] because formats for eager bisimulation [47,48,52,53] are simpler than those for the standard weak bisimulation [13]. Moreover, since the formats for eager bisimulation allow rules with negative premises [47,48] or rules with orderings [52,53], which have the effect of negative premises, they are more general than the formats for weak bisimulation [13]. Another reason for choosing eager bisimulation is that, unlike eager bisimulation, weak bisimulation is not preserved by some simple and useful operators (and the problem is not due to the initial silent actions). For example, the action refinement operator [53] preserves eager bisimulation but not weak bisimulation. Having said that, we agree that both bisimulation relations are equally suitable for all those process languages which are defined by rules with no negative premises and where the divergence is not considered.

We say that an n-ary process operator f preserves a preorder $\sqsubseteq$ if $p_i \sqsubseteq q_i$, for $1 \leqslant i \leqslant n$, implies $f(p_1, \ldots, p_n) \sqsubseteq f(q_1, \ldots, q_n)$ for all processes p_i and q_i. A process language preserves $\sqsubseteq$ if all its operators preserve $\sqsubseteq$. It is well known that eager bisimulation and many other weak process relations are not preserved by some popular process operators, for example the CCS +. We have $a \sqsubseteq_{\approx} \tau.a$ but not $a + b \sqsubseteq_{\approx} \tau.a + b$. The standard solution to this problem is to use the *rooted* version of the preferred relation instead the relation itself: [33] and Definition 5.8.1 in [11]. In our case we will use the *rooted eager bisimulation* relation.

Definition 4. Given $(\mathcal{P}, \mathsf{Act}_\sigma, \rightarrow)$, a relation $R \subseteq \mathcal{P} \times \mathcal{P}$ is a *rooted* eager bisimulation if, for all $(p, q) \in R$, the following properties hold.

$(R.a)$ $\forall\chi.\ (p \xrightarrow{\chi} p'$ implies $\exists q', q''.\ (q \stackrel{\tau}{\Rightarrow} q' \xrightarrow{\chi} q''$ and $p' \sqsubseteq_{\approx} q''))$

$(R.c)$ $p\Downarrow$ implies $\forall\chi.\ (q \xrightarrow{\chi} q'$ implies $\exists p', p''.\ (p \stackrel{\tau}{\Rightarrow} p' \xrightarrow{\chi} p''$ and $p'' \sqsubseteq_{\approx} q'))$

$p \sqsubseteq_{\approx_r} q$ if there exists a rooted eager bisimulation R such that pRq.

It is easy to show that $\sqsubseteq_{\approx_r}$ is a preorder. One may wonder why in the above definition we did not include a property corresponding to (b) of Definition 2. It is because when $p \sqsubseteq_{\approx_r} q$ then $p\Downarrow$ implies $q\Downarrow$.

In Section 3 and afterwards we shall use action σ solely to represent a synchronous passage of time. We will extend the definitions of process operators by specifying how the

timed behaviour of processes depends on the timed behaviour of their components. Consequently, $\sqsubseteq_r$ will not be totally appropriate for timed processes: there are timed versions of standard operators, for example the timed version of the choice operator + of CCS, which do not preserve $\sqsubseteq_r$. Thus, we will use a slight alteration of $\sqsubseteq_r$, called timed rooted eager bisimulation, as our preferred preorder on timed processes.

2.2. GSOS and Ordered SOS formats, and OSOS process languages

Definition 5. A *format* is a SOS rule-based method for assigning operational meaning to process operators.

The first format was proposed by de Simone [46]. de Simone rules are GSOS rules (see Definition 6) without negative premises and copying. The GSOS [14,15] format is due to Bloom et al. [14,15]. The first author developed the ISOS format [47,48], 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 [25,22] and the *panth* format [56].

One of the main motivations for the development of Ordered SOS format [52,53] was to assist the search for formats of SOS rules that treat different groups of actions according to their intended meaning. So far, we have considered

- visible and silent actions and proposed formats [53,55] for several weak process semantics ranging from testing preorder [36], through refusal simulation preorder [47] to eager and branching bisimulation preorders,
- untimed and timed actions as well as visible and silent actions [54]: the present paper is the continuation of this research.

We rejected the *lookahead* feature (process variables on the right-hand side of some transitions coincide with process variables on the left-hand side of other transitions in the premises of rules) as it is unsafe for weak semantics [13,18,47]. As can be seen in the work of Bloom [13] and in [47,48], it is not straightforward to accommodate safely visible and silent actions in SOS rules with negative premises. In order to understand this problem better and derive a solution the OSOS format was developed as an alternative to GSOS. Instead of negative premises OSOS employs orderings on rules that specify the order of their application when deriving transitions of process terms. This new feature allowed us to formulate how negative information can be used safely in rules with visible and silent actions. We achieved this with three simple global conditions on the rule orderings, namely conditions (5)–(7) in Fig. 1, as opposed to seemingly arbitrary restrictions on the form of negative premises as in [47,48]. The conditions are quite intuitive, and they explain the syntactic restrictions on negative premises arrived at in [47,48]; further details can found in Section 2.4 and in [53]. With regard to the present work, we find the use of orderings on rules and global conditions on orderings a convenient and intuitive method to ensure that our process languages with discrete relative time preserve certain timed properties such as maximal progress.

Before we recall the definitions of the formats we introduce several notions and notations.

Var is a countable set of variables ranged over by $X, X_i, Y, Y_i, \ldots$. Σ_n is a set of operators with arity n. A signature Σ is a collection of all Σ_n and it is ranged over by $f, g, \ldots$.

The members of Σ_0 are called *constants*; "**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 $T(\Sigma)$, 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 $\mathbb{T}(\Sigma, \{X_1, \ldots, X_n\})$, where all X_i are distinct. 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 $1 \leqslant i \leqslant n$.

We will use bold italic font to abbreviate the notation for sequences. For example, a sequence of process terms $p_1, \ldots, p_n$, for any $n \in \mathbb{N}$, will often be written as $\boldsymbol{p}$ when the length is understood from the context. Given any binary relation R on closed terms and $\boldsymbol{p}$ and $\boldsymbol{q}$ of length n, we will write $\boldsymbol{p}R\boldsymbol{q}$ to mean $p_i R q_i$ for all $1 \leqslant i \leqslant n$. Moreover, instead of $f(X_1, \ldots, X_n)$ we will often write $f(\boldsymbol{X})$ when the arity of f is understood. An equivalence relation $\approx$ for a process language over Σ is a *congruence* if $\boldsymbol{p} \approx \boldsymbol{q}$ implies $C[\boldsymbol{p}] \approx C[\boldsymbol{q}]$ for all $\boldsymbol{p}$ and $\boldsymbol{q}$ of length n and all Σ contexts $C[\boldsymbol{X}]$ with n holes.

A *closed substitution* is a mapping $\mathsf{Var} \to T(\Sigma)$. Closed substitutions are ranged over by ρ, ρ' and σ; they extend to $\mathbb{T}(\Sigma, \mathsf{Var}) \to T(\Sigma)$ 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 $1 \leqslant i \leqslant n$.

Definition 6 [15]. A GSOS rule is an expression of the form

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

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, and they may appear zero, once or more than once in $C[\boldsymbol{X}, \boldsymbol{Y}]$. 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.

The GSOS format consists of GSOS rules.

Next, we define several notions naming the components of GSOS rules, and SOS rules in general.

Definition 7. Let r be the above GSOS rule. Operator f is the *operator* of r and $rules(f)$ is the set of all rules with the operator f. Expressions $t \xrightarrow{\chi_{ij}} t'$ and $t \overset{\chi_{kl}}{\not\to}$, where $t, t' \in \mathbb{T}(\Sigma, V)$, are called *transitions* and *negative transitions* respectively. Transitions are ranged over by T and T'. If transition T is $X \xrightarrow{a} X'$, we will sometime write $\neg T$ to denote $X \overset{a}{\not\to}$. A (negative) transition which involves only closed terms is called a *closed* (negative) transition. The set of transitions and negative transitions above the horizontal bar in r is called the *premises* of r, and is written as $pre(r)$. The transition below the bar in r is the *conclusion*, written as $con(r)$. Action χ in the conclusion of r is the *action* of r, written as $act(r)$, and $f(\boldsymbol{X})$ and $C[\boldsymbol{X}, \boldsymbol{Y}]$ are the *source* and *target* of r, respectively.

The set of all actions χ_{ij} in the premises of r is denoted by $actions(r)$, and $actions(r, i)$ is the set $\{\chi_{ij} \mid j \in J_i\}$. If $act(r) = \sigma$ and $actions(r) \subseteq \{\sigma\}$, then r is a *timed* rule. If $act(r) = \tau$ and $actions(r) = \{\tau\}$, then r is a *silent* rule. A rule is a *visible action* rule, or simply action rule, if $\tau, \sigma \notin actions(r)$ and $act(r) \neq \sigma$.

The ith argument X_i is *active* in r if r has a positive or a negative premise for X_i. The set of all i such that X_i is active in r is denoted by $active(r)$. The ith argument of f is *active* if $i \in active(r')$ for some rule r' for f.

Example 8. The two most popular operators that are usually defined by rules with negative premises are the ACP priority operator θ [8] and the Bloom's version of sequential composition operator ";" [15]. For a given irreflexive partial order $\ll$ on actions process $\theta(P)$ is a restriction of P such that, in any state of P, action a can happen only if no action b with $a \ll b$ is possible in that state. For each action a we calculate $B_a = \{b \mid a \ll b\}$, and define θ by the following GSOS rules, one for each action a and set B_a.

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

The sequential composition operator is defined by two GSOS rule schemas, for any visible a and c:

$$\frac{X \xrightarrow{a} X'}{X; Y \xrightarrow{a} X'; Y}\, r'_{a\star}, \qquad \frac{Y \xrightarrow{c} Y' \quad \{X \overset{a}{\not\to} \mid \text{all } a\}}{X; Y \xrightarrow{c} Y'}\, r'_{\star c}.$$

In general, the behaviour of a process derived by a rule r with a negative premise $X_i \overset{a}{\not\to}$ depends on the inability of the ith component to perform a. What does happen when the component is able to perform a, and indeed performs a, when tested for the refusal of a? Typically, for all (known to us) popular process operators that are defined by rules with negative premises, there are other rules with the premise $X_i \xrightarrow{a} X'_i$. Informally, these rules have in some sense higher priority than r: when the ith component can perform a such rules are potentially applicable, and r is certainly not applicable. This suggests that SOS rules may be applied in a certain order when deriving transitions of process terms, so r is applicable when no rules higher according to the order are applicable. Hence, a rule r with the negative premise $X_i \overset{a}{\not\to}$ is equivalent, in some sense, to r with $X_i \overset{a}{\not\to}$ removed being placed below a rule with the premise $X_i \xrightarrow{a} X'_i$. So, it should be possible to remove all negative premises from rules and mimic them by an ordering on the resulting rules. This simple approach works for all popular operators, but in general some encoding is required. The definitions of rules with orderings, the resulting formats and process languages are given below, as well as many illustrating examples.

Definition 9. A *positive* GSOS rule (OSOS rule, or simply a transition rule) is a GSOS rule with $K = \emptyset$. With the notation as in Definition 6, it has the form

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

Next, we recall the notion of *ordering* on rules [53]. This new feature allows the user to control the order of application of OSOS rules when deriving transitions of process terms.

An ordering on OSOS rules for operator f, $<_f$, is a binary relation over the rules for f. In a large majority of cases the ordering relations are *irreflexive* (i.e. $r < r$ never holds) and *transitive*. In general, however, there are operators, which are described and motivated in [53], which use non-transitive or not irreflexive orderings on rules: see example 16 below. Expression $r' <_f r$ is interpreted as r having higher priority than r' when deriving

transitions of terms with f as the outermost operator. Given Σ, the relation $<_\Sigma$, or simply $<$ if Σ is known from the context, is defined as $\bigcup_{f\in\Sigma} <_f$. We will denote $\{r' \mid r < r'\}$ as $higher(r)$.

Definition 10. The *OSOS format* consists of OSOS rules equipped with orderings on rules.

Now, we define the notion of a process language. A process language is given by specifying its operators and by a "method" for defining the operational meaning of the operators. Often, such method is just a set of SOS rules but, in our case, it can also be a set of SOS rules equipped with an ordering. Hence, the following informal definition.

Definition 11. A process language is a tuple (Σ, A, M), where Σ is a finite set of operators, $A \subseteq \mathsf{Act}_\sigma$ and M is a method for giving operational meaning of the elements in Σ.

For the ease of use we shall state explicitly the method M for each process language that we consider. This will require a slight overloading of the notion of a process language: it will be either a triple or quadruple depending whether method M is just a set of SOS rules or a set of rules equipped with an ordering.

Definition 12. A GSOS process language is a tuple (Σ, A, R), where Σ is a finite set of operators, $A \subseteq \mathsf{Act}_\sigma$, R is a finite set of GSOS rules for operators in Σ such that all actions mentioned in the rules belong to A. An operator is GSOS if its meaning is given by rules in the GSOS format.

An *Ordered SOS* (or OSOS, for short) process language is a tuple $(\Sigma, A, R, <)$, where Σ is a finite set of operators, $A \subseteq \mathsf{Act}_\sigma$, R is a finite set of rules for operators in Σ, written as $rules(\Sigma)$, such that all actions mentioned in the rules belong to A, and $<$ is an ordering on $rules(\Sigma)$. An operator is OSOS if its meaning is given by rules in the OSOS format.

Given an OSOS process language $G = (\Sigma, A, R, <)$, we associate a unique transition relation $\rightarrow$ with G. Full details are given in [53]; we only recall the basics. Let $d : T(\Sigma) \to \mathbb{N}$ 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 $d(f(p_1, \ldots, p_n)) = 1 + \max\{d(p_i) \mid 1 \leqslant i \leqslant n\}$ otherwise.

Definition 13. Given an OSOS process language $(\Sigma, A, R, <)$, we associate with it a transition relation, $\rightarrow \subseteq T(\Sigma) \times A \times T(\Sigma)$, which is defined as $\rightarrow = \bigcup_{l<\omega} \rightarrow^l$, where transition relations $\rightarrow^l \subseteq T(\Sigma) \times A \times T(\Sigma)$ are

$$
\begin{array}{lll}
p \xrightarrow{\chi} p' \in \rightarrow^l & \text{iff} & d(p) = l \text{ and} \\
 & & \exists r \in R,\ \exists \rho. \quad (\rho(con(r)) = p \xrightarrow{\chi} p' \text{ and} \\
 & & \qquad \rho(pre(r)) \subseteq \bigcup_{k<l} \rightarrow^k \text{ and} \\
 & & \qquad \forall r' \in higher(r).\ \rho(pre(r')) \not\subseteq \bigcup_{k<l} \rightarrow^k).
\end{array}
$$

The definition states that a rule r can be used to derive a transition $p \xrightarrow{\chi} p'$ if r is enabled at p, i.e. $p \xrightarrow{\chi} p'$ coincides with the conclusion of r under some 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 14. Let $r \in rules(f)$ and $pre(r) = \{X_i \xrightarrow{\chi_{ij}} Y_{ij} \mid i \in I, j \in J_i\}$. Rule r *applies* to $f(\boldsymbol{u})$ if and only if $u_i \xrightarrow{\chi_{ij}}$ 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.

Having defined the transition relation for G, the LTS for G is $(T(\Sigma), A, \rightarrow)$. Thus, eager bisimulation, its rooted and timed rooted versions are defined over this LTS as in Definitions 2, 4 and 32 (Section 3.4), respectively.

Example 15. Operational definitions in terms of OSOS rules with orderings can be given alternatively in terms of GSOS rules with negative premises. Consider the sequential composition operator ; from Example 8. It can be defined by simple OSOS rules

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

together with the ordering $<$ on rules defined by $r_{*c} < r_{a*}$ for all a and c. Note that by the GSOS rules $p; q$ can perform an initial action of q if $p \xrightarrow{a}$ for all a. Using the OSOS rules, $p; q$ can perform an initial action of q if rule r_{*c} is enabled at $p; q$. This is when r_{*c} applies to $p; q$, and all rules r_{a*} (i.e. rules higher than r_{*c}) do not apply. The second condition means that $p \xrightarrow{a}$ for all a.

The priority operator θ from Example 8 can be defined equivalently by OSOS rules equipped with an ordering to represent the priority order on actions. The rules for the OSOS version of θ are, one for each a,

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

and the ordering $<$ is $r_a < r_b$ for all a and $b \in B_a$, i.e. for all b such that $a \ll b$. The ordering prescribes that rule r_a can be applied to derive a transition of $\theta(P)$ if no rule with higher priority, e.g. r_b, can be applied to $\theta(P)$.

2.3. GSOS = OSOS

The expressiveness results in [53] show that in general OSOS rules with orderings have the same effect as GSOS rules with negative premises. For the illustration we recall two methods for reformulating arbitrary GSOS process languages as OSOS process languages. Also, we explain briefly how arbitrary OSOS process languages can be redefined as GSOS process languages. We give an example of a popular operator whose OSOS formulation is superior to its GSOS formulation.

When we say that a process language can be reformulated, or equivalently expressed, as another process language, we mean that the two process languages determine the same LTS. Often it is more convenient to discuss individual operators or groups of operators rather than whole process languages, for example in Section 5 or in Example 15. In the latter, we consider the GSOS and OSOS versions of ; and θ. In such cases we implicitly assume that each version of the operator is a part of a simple process language that also

contains the CCS prefixing, + and **0** (or their time preserving extensions). Note that these operators are both GSOS and OSOS. Hence, the GSOS ;, the prefixing, + and **0** form a GSOS process language, the OSOS version of ; together with the last three operators form an OSOS process language, and LTSs for these process languages can be computed and compared.

Most of the popular process operators that are defined (or may be defined) by rules with negative premises (GSOS rules or *ntyft/ntyxt* rules [16,22]) have OSOS formulations which are as natural and efficient as those of the sequential composition and the priority operators discussed above. To mention just few such operators we have the priority choice in [51], action refinement operator [53], hide operator of ET-LOTOS [31], several delay operators [5,9,27,39], and several timed extensions of traditional operators: parallel composition of *TPL* (Example 17), and hiding and sequential composition of CSP with time [43,45].

Not all OSOS formulations of general GSOS operators (although we cannot think of a widely used such operator) are so neat, where a GSOS operator is equivalently expressed as an OSOS operator. Some contrived GSOS operators cannot be expressed as OSOS operators but only as simple OSOS terms. For completeness the following example illustrates two methods, introduced in [53], for reformulating GSOS operators in the OSOS format: the first uses orderings which are arbitrary relations, for example not irreflexive, and the second uses irreflexive and transitive orderings. Hence, the second method shows that arbitrary GSOS operators, no matter how unusual, have pleasant and in intuitive formulations in terms of OSOS simple terms.

Example 16. Let a GSOS operator g be defined by a single rule r below.

$$\frac{X \xrightarrow{a} X' \quad Y \overset{b}{\not\to}}{g(X,Y) \xrightarrow{a} \mathbf{0}}.$$

As g has no rule with the premise $Y \xrightarrow{b} Y'$, we see no other way to redefine g but to use auxiliary rules and appropriate orderings. We give two different definitions of g within the OSOS format. Firstly, consider the rules

$$\frac{X \xrightarrow{a} X'}{g(X,Y) \xrightarrow{a} \mathbf{0}}\, r_{a*}, \qquad \frac{Y \xrightarrow{b} Y'}{g(X,Y) \xrightarrow{b} \mathbf{0}}\, r_{*b},$$

which are ordered by $r_{a*} < r_{*b}$. If $q \overset{b}{\not\to}$ and $p \xrightarrow{a}$, then it is clear that $g(p,q) \xrightarrow{a}$ by rule r_{a*}. But, when $q \xrightarrow{b}$, we obtain $g(p,q) \xrightarrow{b}$ by r_{*b}. This transition cannot be derived using the GSOS definition. To stop this, we impose $r_{*b} < r_{*b}$ which makes r_{*b} never enabled at $g(p,q)$. Notice that the ordering is not irreflexive.

Secondly, we shall use an auxiliary *restriction* operator instead of the auxiliary rules that are above themselves and thus never enabled. Operator g is defined by r_{a*} above and the rule r_{*error} below. The rules for the restriction operator "$\backslash error$", one for each a in Act, are given below, where *error* is a new action not in Act.

$$\frac{Y \xrightarrow{b} Y'}{g(X,Y) \xrightarrow{error} \mathbf{0}}\, r_{*error}, \qquad \frac{X \xrightarrow{a} Y}{X \setminus error \xrightarrow{a} Y \setminus error}.$$

The ordering on rules for g is $r_{a*} < r_{*error}$. Moreover, the ordering is irreflexive and transitive. If $q \xrightarrow{b}$, we obtain $g(p,q) \xrightarrow{error}$ by r_{*error}. To prevent this, we apply $\backslash error$ at

the outermost level. Hence, $g(X, Y)$ is translated to $g(X, Y) \setminus error$. It can be checked that the process $g(p, q) \setminus error$ has the same behaviour as the original GSOS process $g(p, q)$.

Next, we recall a method from [53] for reformulating OSOS operators in terms of GSOS rules. Suppose that f is an n-ary OSOS operator with set of ordered rules R_f. For each $r \in R_f$ we show how to construct the set $R'_f(r)$ of GSOS rules for f which has the same effect as $\{r\} \cup higher(r)$. Then, the set R'_f of GSOS rules for f is $\bigcup_{r \in R_f} R'_f(r)$. If $higher(r) = \emptyset$, then clearly $R'_f(r) = \{r\}$. Otherwise, assume $higher(r) = \{r_k \mid 1 \leqslant k \leqslant m\}$. If one of the rules in $higher(r)$, say r', has no premises, then rule r is never enabled because r' is always applicable. Thus, $R'_f(r) = \emptyset$. If none of the rules in $higher(r)$ has empty premises, then $R'_f(r)$ consists of GSOS rules of the following form.

$$\frac{pre(r) \cup \{\neg T_k \mid k \in \{1, \ldots, m\} \ \wedge \ T_k \in pre(r_k)\}}{con(r)}.$$

The set of negative premises in any rule in $R'_f(r)$ consists of m negative transitions $\neg T_k$, where each T_k is one of the (positive) premises in $r_k \in higher(r)$. We easily calculate that $R'_f(r)$ has $\prod_{k=1}^{m} n_k$ rules, where $n_k \geqslant 1$ is the number premises in $r_k \in higher(r)$. 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 $higher(r)$ is more concise than the corresponding fragment $R'_f(r)$.

Example 17. Consider Hennessy and Regan's *Temporal Process Language* (TPL) [27]. It has a delay operator "$\lfloor\ \rfloor(\ )$" defined by the following GSOS rules, where a is any visible action and the action σ denotes the passage of one time unit.

$$\frac{X \xrightarrow{a} X'}{\lfloor X \rfloor(Y) \xrightarrow{a} X'}, \qquad \frac{X \xrightarrow{\tau} X'}{\lfloor X \rfloor(Y) \xrightarrow{\tau} X'}, \qquad \frac{X \xrightarrow{\tau}}{\lfloor X \rfloor(Y) \xrightarrow{\sigma} Y}.$$

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

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

and the ordering is $\sigma_\emptyset < \tau^1$. The parallel composition operator "$\|$" of TPL is an extension of the CCS parallel with the following (non-GSOS) ntyft rule.

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

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

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

which is placed below all the rules for $\|$ with the action τ, namely the two τ-rules and all the communications rules $r_{a\bar{a}}$. The GSOS formulation of the operator, following the method described above, consists of the standard CCS rules together with the rules of the form

$$\frac{X \xrightarrow{\sigma} X' \quad Y \xrightarrow{\sigma} Y' \quad X \overset{\tau}{\not\rightarrow} \quad Y \overset{\tau}{\not\rightarrow} \quad \{\neg T_a \mid a \in \mathsf{Act} \setminus \{\tau\} \ \wedge \ T_a \in pre(r_{a\overline{a}})\}}{X \parallel Y \xrightarrow{\sigma} X' \parallel Y'},$$

where the premise $\neg T_a$ for a fixed a is either $X \overset{a}{\not\rightarrow}$ or $Y \overset{\overline{a}}{\not\rightarrow}$. Comparing the two definitions, we note that the OSOS formulation uses one timed rule r_σ and the ordering which places r_σ below all the rules for $\parallel$ with the action τ, whereas the GSOS formulation uses 2^k timed rules, where $k = |\ \mathsf{Act} \setminus \{\tau, \sigma\}\ |$.

2.4. OSOS process languages for rooted eager bisimulation

Finally, we recall the definition of a subclass of OSOS process languages whose operators preserve $\overset{\sqsubseteq}{\sim}_r$. We define several conditions that restrict the structure of OSOS rules and the orderings on such rules, and show that if OSOS operators satisfy these conditions, then they preserve $\overset{\sqsubseteq}{\sim}_r$. We shall need the following notions.

Among the transition rules we shall be particularly interested in rules which describe the unobservable behaviour of processes in terms of the unobservable behaviour of their components. We shall call such rules the *silent rules*: a rule r is a silent rule if $act(r) = \tau$ and $actions(r) = \{\tau\}$. The most widely used silent rules are τ-rules, also called *patience* rules, [12]. The τ-rule for the ith argument of f, denoted by τ_i^f or simply by τ_i if f is clear from the context, has the form

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

The set of τ-rules for a rule r for f (or for f itself) consists of all τ_i such that $i \in active(r)$ (or $i \in active(f)$).

The second form of silent rules we shall consider are the *silent choice rules*. The silent choice rule for the ith argument of f, denoted by $\tau^{f,i}$, simply by τ^i if f is clear from the context, has the form

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

Notation. We shall write $tau^f(i)$, or simply $tau(i)$ if f is clear from the context, to denote either τ_i^f or $\tau^{f,i}$. Also, in several conditions in Section 4 where f is clear from the context, we shall write $silent(r)$ to mean that rule r is either a τ-rule or a silent choice rule for one of the active arguments of f.

A silent choice rule is an instance of a more general form of rules called *choice rules*. Given n-ary operator f, a rule is a choice rule for f and i (its ith argument) if it has the following form, where α is any action:

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

Examples of choice rules are the rules for + of CCS, the rule for the second argument of; and the first two rules for the delay operator in Example 17. An example of operator with silent rules that we do not cover in this paper is the ACP left-merge.

Certain occurrences of process variables in rules are called *copies*. They can be divided into *explicit* and *implicit* copies [48,53]:

Definition 18. Given a rule r as in Definition 9, the explicit copies are the multiple occurrences of variables Y_{ij} in $C[\boldsymbol{X}, \boldsymbol{Y}]$ and the multiple occurrences of variables X_i, for $i \notin I$, in $C[\boldsymbol{X}, \boldsymbol{Y}]$. The implicit copies are the multiple occurrences of X_i in the premises of r, i.e. when $\mid J_i \mid > 1$, and the occurrences, not necessarily multiple, of variables X_i in $C[\boldsymbol{X}, \boldsymbol{Y}]$ when $i \in I$. The set of all implicit copies of process variables in a rule r for a given operator f is denoted by *implicit-copies*(r).

Consider the following rule r_h:

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

This rule is a recipe for working out all the a-derivatives of process terms of the form $h(p_1, p_2, p_3, p_4)$. In order to derive the a transition of $h(p_1, p_2, p_3, p_4)$, 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 X_1 and X_2 are 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 X_3 and Y_{11} are explicit. The transition a is the cost of making these copies, whereas copies of X_1 and X_2 are there for 'free'. It can be seen in [48,53] 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.

We are ready to recall the definition of a general class of OSOS process languages whose operators preserve $\sqsubseteq_r$. The class is called *rooted eager bisimulation ordered* or rebo for short. Consider conditions in Fig. 1. The conditions are for an arbitrary OSOS operator f, $<$ is the ordering on the rules for f, r and r' range over *rules*(f). $\tau(i)$ stands for either τ_i or τ^i, which are the τ-rule and silent choice rule for the ith argument of f, respectively. All the conditions are (implicitly) universally quantified over i, and r and r' where appropriate. The conditions are a slight modification of those that appeared in [55].

Definition 19. Let f be an OSOS with $active(f) \neq \emptyset$. The operator f is τ-*preserving* if the set of its rules and the ordering on the rules satisfy (1)–(8). The operator f is τ-*sensitive* if it has a silent choice rule for one of its active arguments and the set of its rules and the ordering on the rules satisfy (1)–(2) and (4)–(8).

An OSOS process language is rooted eager bisimulation ordered (rebo) if its signature can be partitioned into three groups: operators with no active arguments, τ-preserving operators and τ-sensitive operators, and the targets of all rules, except for the τ-rules of τ-sensitive operators, do not contain τ-sensitive operators. Operators of rebo process languages are called rebo operators.

if $\tau \in actions(r, i)$ **then** $r = tau(i)$ (1)

if $i \in active(f)$ **then** $tau(i) \in rules(f)$ **and**
(**either** $tau(i) = \tau_i$ **or** $tau(i) = \tau^i$) (2)

if $i \in active(f)$ **then** $tau(i) = \tau_i$ (3)

if $i \in active(r)$ **and** $tau(i) = \tau^i$ **then** r is a choice rule (4)

not $(tau(i) < tau(i))$ (5)

if $r' < r$ **and** $i \in active(r)$ **then** $r' < tau(i)$ (6)

if $tau(i) < r$ **and** $i \in active(r') \cup active(higher(r'))$ **then** $r' < r$ (7)

if $i \in implicit\text{-}copies(r)$ **then** $r < tau(i)$ (8)

Fig. 1. Conditions for τ-preserving operators and τ-sensitive operators.

The τ-rules for any operator f that are guaranteed by conditions (2)–(3) shall be henceforth called the τ-rules *associated with* f. Correspondingly, given any rule r for f, all τ-rules τ_i for f such that $i \in active(r)$ shall be called the τ-rules *associated with* r. More generally, using condition (2), we define the silent rules *associated with* f and the silent rules *associated with* r for f in the corresponding fashion.

The conditions in Fig. 1 can be grouped into the conditions on the structure of rules: (1)–(4), and the conditions on the orderings on rules: (5)–(8). Condition (1) requires that τ-rules or silent choice rules are the only rules with actions τ in the premises. Condition (2) insists that for each active argument of an operator there is a silent rule for that argument among its rules, and that silent rule is **either** a silent choice rules **or** a τ-rules. If the silent rule for active ith argument of a rule is a silent choice rule, then (4) insists that the rule itself must be a choice rule. We group operators into τ-preserving and τ-sensitive operators. We require that silent rules associated with τ-preserving operators are purely τ-rules (condition (3)), and silent rules associated with τ-sensitive operators could be either τ-rules or silent choice rules: (2).

Remark 20. In the rest of the paper, when we say silent rules we shall mean, due to the above conditions, either τ-rules or silent choice rules.

Condition (5) insists that silent rules are always enabled: putting a silent rule below itself will disable it forever, thus is disallowed. The intuition for (6) is that before we apply r' we must check that all rules with higher priority, and thus their associated silent rules, are not applicable. Condition (7) requires that if a rule disables a silent rule for argument i, then it also disables all rules with active i (call this set R), and all other rules with active i that are below the rules in R. Finally, (8) allows only implicit copies of stable arguments.

Conditions in Fig. 1 were introduced in [53,55] to help to define subclasses of OSOS process languages that preserve divergence sensitive versions of eager bisimulation and branching bisimulation, and their rooted versions. Examples were also given in the above

references that demonstrate that the conditions could not be relaxed without compromising the relevant congruence results, except perhaps for allowing some other types of silent rules. Because of the importance of (5)–(7) we present several examples, based on the examples in [53], which show that these conditions cannot be dropped or relaxed.

A careless ordering on rules may change the unobservable and the independent of the environment 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.

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

The ordering is defined by $\tau_2 < r_{a*}$ for all actions a. Consider rooted eager bisimulation (or rooted branching bisimulation) 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 $c.\mathbf{0} \parallel a.\tau.b.\mathbf{0} \xrightarrow{a} c.\mathbf{0} \parallel \tau.b.\mathbf{0} \xrightarrow{\tau}$ since $\tau_2 < r_{c*}$. In order to prevent such orderings we might require

if r is a τ-rule then $higher(r) = \emptyset$.

The intuition here 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 condition might be as follows:

if $\tau_i < r$ and $i \in active(r')$ then $r' < r$.

When orderings are transitive relations, the above condition is what is needed. In general, however, the condition is too weak. This is best illustrated by the following example: Consider operators g and f defined by the following rules, where a and c are particular actions in Vis,

$$\frac{Y \xrightarrow{a} Y'}{g(X, Y) \xrightarrow{a} f(X, Y')}, \qquad \frac{Y \xrightarrow{\tau} Y'}{g(X, Y) \xrightarrow{c} g(X, Y')},$$

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

together with the associated τ-rules for f which we denote by τ_1 and τ_2. There is no ordering on the rules for g, and the ordering on the rules for f is as follows. Note that the ordering is *not* transitive.

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

We easily check that $<$ satisfies the above condition. Now consider rooted eager bisimilar processes $a.\mathbf{0}$ and $a.\tau.\mathbf{0}$, and the processes $g(a.\mathbf{0}, a.\mathbf{0})$ and $g(a.\mathbf{0}, a.\tau.\mathbf{0})$. We have $g(a.\mathbf{0}, a.\mathbf{0}) \xrightarrow{a} f(a.\mathbf{0}, \mathbf{0})$ and $f(a.\mathbf{0}, \mathbf{0}) \xrightarrow{a}$ by r_a since r_c and τ_2 are not applicable as

$\mathbf{0} \stackrel{\tau}{\not\to} \stackrel{c}{\not\to}$. But, after $g(a.\mathbf{0}, a.\tau.\mathbf{0}) \stackrel{a}{\to} f(a.\mathbf{0}, \tau.\mathbf{0})$ we have $f(a.\mathbf{0}, \tau.\mathbf{0}) \stackrel{a}{\not\to}$ since $r_a < \tau_2$ and τ_2 is applicable. Also, $f(a.\mathbf{0}, \tau.\mathbf{0}) \stackrel{\tau}{\not\to}$ since $\tau_2 < r_a$ and r_a is applicable. Hence, although $a.\mathbf{0}$ and $a.\tau.\mathbf{0}$ are equivalent the processes $g(a.\mathbf{0}, a.\mathbf{0})$ and $g(a.\mathbf{0}, a.\tau.\mathbf{0})$ are not.

Since $\tau_2 < r_a$ we expect, by the above condition, r_a to be above all rules with active second argument. In order to cover for the lack of transitivity, it is also reasonable to expect r_a to be above all rules which are below some 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 the ordering is transitive. However, we do not assume transitivity in general, and so we strengthen the condition as follows, and thus obtain condition (7), where $tau(i)$ is either τ_i or τ^i:

if $tau(i) < r$ and $i \in active(r') \cup active(higher(r'))$ then $r' < r$.

Notice that (7) implies the following limited form of transitivity:

if $r' < tau(i) < r$ then $r' < r$.

Returning to the operator f above, since $\tau_2 < r_a$ and $2 \in active(higher(r_a))$ we deduce $r_a < r_a$ by (7). Thus, $f(a.\mathbf{0}, \mathbf{0}) \stackrel{a}{\not\to}$, and the pairs of processes $f(a.\mathbf{0}, \mathbf{0})$ and $f(a.\mathbf{0}, \tau.\mathbf{0})$, and $g(a.\mathbf{0}, a.\mathbf{0})$ and $g(a.\mathbf{0}, a.\tau.\mathbf{0})$ are equivalent.

Unlike our first attempt, condition (7) does not prohibit a τ-rule to be above itself. In order to see that this is problematic, consider the CCS-like renaming operator "$[R]$" which renames a by b. The ordering on its rules satisfies $\tau_1 < \tau_1$ and condition (7). Thus, τ_1 is above all action rules for $[R]$. As a result, although $a.a.\mathbf{0}$ and $a.\tau.a.\mathbf{0}$ are rooted eager bisimulation equivalent the processes $(a.\mathbf{0})[R]$ and $(\tau.a.\mathbf{0})[R]$ are not: $(a.a.\mathbf{0})[R] \stackrel{b}{\to} (a.\mathbf{0})[R] \stackrel{b}{\to}$ but $(a.\tau.a.\mathbf{0})[R] \stackrel{b}{\to} (\tau.a.\mathbf{0})[R] \stackrel{\tau}{\not\to}$ 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 (5) which requires that τ-rules and silent choice rules are never disabled.

not $(tau(i) < tau(i))$.

There are operators which are definable by ordered rules satisfying conditions (7) and (5) but which are not well behaved. Consider the priority operator θ from Example 15 which gives b priority over a, i.e. $a \ll b$, hence $r_a < r_b$. Let $p = a.\mathbf{0} \parallel b.\tau.b.\mathbf{0}$ and $q = a.\mathbf{0} \parallel b.b.\mathbf{0}$, where $\parallel$ is the CCS parallel. Clearly, p and q are rooted eager bisimulation equivalent. However, we have $\theta(p) \stackrel{b}{\to} \theta(a.\mathbf{0} \parallel \tau.b.\mathbf{0}) \stackrel{a}{\to} \theta(\mathbf{0} \parallel \tau.b.\mathbf{0})$ since r_a is not disabled by τ_1, and $\theta(q) \stackrel{b}{\to} \theta(a.\mathbf{0} \parallel b.\mathbf{0}) \stackrel{a}{\not\Rightarrow}$. In order to repair this it is enough to require $r_a < \tau_1$, i.e. that silent rules for active arguments of rules higher than a rule (r_a) must also be above the rule. This is condition (6):

if $r' < r$ and $i \in active(r)$ then $r' < tau(i)$.

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

Example 21. In the setting with silent actions the rebo definition of the sequential composition ; is obtained by adding to the OSOS definition from Example 15 two silent rules below and ordering them appropriately.

$$\frac{X \stackrel{\tau}{\to} X'}{X; Y \stackrel{\tau}{\to} X'; Y}\ \tau_1, \qquad \frac{Y \stackrel{\tau}{\to} Y'}{X; Y \stackrel{\tau}{\to} Y'}\ \tau^2.$$

The first rule is a τ-rule and the second is a silent choice rule. The ordering is extended to include $\tau^2 < \tau_1$. Also, it must satisfy the conditions of Definition 19: $\tau^2 < r_{a*}$ for all a by (7), and $r_{*c} < \tau_1$ for all c by (6).

rebo process languages are descendants of process languages for eager bisimulation preorder introduced in [52,53]. The idea of partitioning process operators into two groups, where one of the groups contains τ-sensitive operators, i.e. operators which have no τ-rules, and using the rooted versions of weak equivalences for process languages with such operators is due to Bloom [13]. It was employed by Fokkink in [18], and was used in [53]. Most of the popular process operators are rebo: most of the operators are τ-preserving but some are τ-sensitive. The second group contains the CCS +, our version of ;, the Kleene star [4,3], delay operator in Example 31, Milner's interrupt operators [33] and others. Two exceptions of popular operators which are not rebo are the ACP left-merge operator and the version of the Kleene star operator in [18]. We claim that the notion of rebo process operators can be slightly generalised by employing the *wild/tame argument* technique due to Bloom and Fokkink, with a result that the two exceptions are in the generalised rebo.

Finally, we present the main result of this section. It permits compositionality of specifications and modularity of verification of systems modelled in rebo process languages. The proof is given in Appendix A.

Theorem 22. *All* rebo *process languages preserve rooted eager bisimulation.*

3. Adding discrete relative time to process languages

In this section we propose a uniform method for extending an arbitrary rebo process language with the notion of discrete relative time so that the resulting process language is time deterministic and it preserves a timed version of rooted eager bisimulation. The extension is achieved in several steps as follows:

1. The basis of a process language with discrete relative time is a rebo process language. The rebo process language contains no actions σ and no timed rules. It consists of a chosen standard process operators and a number of additional new operators that are selected to introduce, alter or stop the passage of time after they are extended with timed rules.
2. We augment the definitions of the standard operators by adding special timed rules and by extending the orderings, if necessary, so that the operators preserve the passage of time. The operators extended in such way shall be called *time preserving* operators.
3. We augment the definitions of the additional operators which are meant to introduce, alter or stop the passage of time by adding general timed rules and extending the orderings if necessary. Such operators shall be called *time altering* operators.

The notion of *priority level* will be used to define the forms of rules for time preserving and time altering operators, hence is introduced below. Then, we define timed process languages and *timed rooted eager bisimulation* relation. Finally, we prove that all timed process languages preserve timed rooted eager bisimulation and that their processes are time deterministic.

3.1. Priority levels

We motivate our method for extending the definitions of rebo operators with discrete relative time as follows: Following [42,38,59], one may expect that, given a rebo operator f, it is sufficient to extend the set of rules for f with the following timed rule, where $J = active(f)$ and $Y_i = X_i$ for $i \notin active(f)$.

$$\frac{\{X_j \xrightarrow{\sigma} Y_j\}_{j\in J}}{f(X_1,\ldots,X_n) \xrightarrow{\sigma} f(Y_1,\ldots,Y_n)}.$$

Notation. A timed rule of the above form, where J is not necessarily the set of active arguments of the operator f, will be called a σ-*rule* and denoted thereafter by σ_J^f. The set J indicates which arguments appear in the premises. We will simply write σ_J when the operator is obvious from the context, and when $J = \{j\}$, for some $j \in \mathbb{N}$, then we simply write σ_j for σ_J. A timed rule which is not a σ-rule may have an arbitrary term in the target of the conclusion.

This extension method works for the + and the parallel composition operators of CCS, and for many other process operators: it guarantees that the time synchrony property holds. But, the method is not suitable for some operators which are defined by ordered rules or, alternatively, by rules with negative premises. For example, the sequential composition operator, which we have defined in Example 15, is given the following rule.

$$\frac{X \xrightarrow{\sigma} X' \quad Y \xrightarrow{\sigma} Y'}{X; Y \xrightarrow{\sigma} X'; Y'}\ \sigma_{\{1,2\}}.$$

The rule requires X and Y to synchronise on σ contrary to the intention that Y should only act when X is deadlocked and cannot pass time. Applying the rule, we deduce that $p; q$ cannot pass time if q cannot pass time, even though p may be able to pass time. This fails time synchrony.

Another reason why rule $\sigma_{\{1,2\}}$ is not suitable is that, when combined with other rules for the sequential composition, it produces a peculiar ordering on the rules. By conditions (7) and (6) we obtain $\sigma_{\{1,2\}} < r_{a\star}$ and $\sigma_{\{1,2\}} < \tau_1$, respectively. As a result, $X; Y$ can only pass time when X cannot perform any visible or silent actions. Our solution will be to use two separate σ-rules, one for each argument of ;, instead of $\sigma_{\{1,2\}}$, and to order them accordingly. In general, an operator will have a timed rule for each of its priority levels.

Condition (2) for an operator f requires that there must be either a τ-rule, τ_i, or a silent choice rule, τ^i, for each active argument X_i of f.

Definition 23. We say that two active arguments X_i and X_j of f have the *same priority* if and only if the silent rules for the arguments are unordered, i.e. $tau(i) \not< tau(j)$ and $tau(j) \not< tau(i)$. If $active(f) \neq \emptyset$, then let $Levels(f)$ be the set of all non-empty subsets K of $active(f)$ such that, for all K, all members of K have the same priority. When $active(f) = \emptyset$, then we set $Levels(f)$ to $\{\emptyset\}$. Members of $Levels(f)$ are called the *priority levels* for f.

For example, the two arguments of the CCS choice and parallel composition operators have the same priority as their τ-rules are unordered with respect to each other. However,

the arguments of ; do not have the same priority since the τ-rule for the first argument is above the silent choice rule for the second argument, i.e. $\tau^2 < \tau_1$.

Note that if $K \in \mathit{Levels}(f)$, then $L \in \mathit{Levels}(f)$ for every non-empty subset L of K. Also, if $K, L \in \mathit{Levels}(f)$ and $K \cap L \neq \emptyset$, then $K \cup L \in \mathit{Levels}(f)$.

Example 24. We have $\mathit{Levels}(+) = \mathit{Levels}(\|) = \{\{1\}, \{2\}, \{1, 2\}\}$ as the orderings on the silent rules for both operators are empty. Since for the sequential composition operator; we have $\tau^2 < \tau_1$, we get $\mathit{Levels}(;) = \{\{1\}, \{2\}\}$.

If L is a priority level, then all its non-empty subsets are also priority levels. A priority level L for f is *maximal* if it is not a proper subset of another priority level for f. For example, $\{1, 2\}$ is a maximal priority level for the CCS $+$ and $\|$ operators. Priority levels $\{1\}$ and $\{2\}$ are not maximal since they are contained in $\{1, 2\}$. However, priority levels $\{1\}$ and $\{2\}$ for the sequential composition operator ; are both maximal.

Next, we define a relation $\prec_f : \mathit{Levels}(f) \times \mathit{Levels}(f)$ as follows: for all $K, L \in \mathit{Levels}(f)$

$$K \prec_f L \quad \text{if} \quad K \neq L \text{ and } \forall k \in K, \exists l \in L.\ (\mathit{tau}(k) < \mathit{tau}(l) \text{ or } k = l).$$

When f is clear from the context, we abbreviate $\prec_f$ by $\prec$. So, $K \prec L$ holds if, informally, the elements of K have lower priority than, or are, the elements of L. For example, consider the priority levels of ;. We have $\{2\} \prec \{1\}$ since $\tau^2 < \tau_1$. For the priority levels of $+$ we have $\{2\} \prec \{1, 2\}$ since $\{2\} \subset \{1, 2\}$. The most crucial use of the relation will be in the definition of σ-rules for time altering operators. If f is such an operator and K and L are its priority levels with $K \prec_f L$, then we will insist that $\sigma_K < \sigma_L$. This requirement will be crucial for the time determinacy property.

We define a few more new notions and notations. Although they are only used later on in Section 4 we introduce them here because they are defined in terms of priority levels. Let $K \downarrow L = \{k \mid k \in K \ \wedge\ \forall l \in L.\ \mathit{tau}(k) \not< \mathit{tau}(l)\}$. In other words, $K \downarrow L$ is the set of all k in K which do not have lower priority than any members of L. Note that when K and L are priority levels and $K \subseteq L$, we have $K \downarrow L = K$ and $L \downarrow K = L$ since all members of L, and thus of K, have the same priority. We have the following straightforward result.

Lemma 25. *If K and L are priority levels, then $M = K \downarrow L \cup L \downarrow K$ is also a priority level.*

Proof. Assume for contradiction that there exist k and l in M such that $\mathit{tau}(k) < \mathit{tau}(l)$. Clearly, both k and l are not members of K, and they are not members of L. Without loss of generality assume that $k \in K$ and $l \in L$. Since $\mathit{tau}(k) < \mathit{tau}(l)$ we deduce $k \notin K \downarrow L$. Hence, $k \notin M$: contradiction. □

Given priority levels K and L, the set $K \downarrow L \cup L \downarrow K$ will be called the *least upper priority level* of K and L, and will be denoted by $\mathsf{lupl}(K, L)$. Clearly, $\mathsf{lupl}(K, L) = \mathsf{lupl}(L, K)$. When K and L are priority levels and $K \subseteq L$, we have $\mathsf{lupl}(K, L) = L$. If $K \subset L$, then $K \prec \mathsf{lupl}(K, L)$ but not $L \prec \mathsf{lupl}(K, L)$. This is because $\mathsf{lupl}(L, K) = L$ and $L \not\prec L$.

Lemma 26. *If K and L are non-empty priority levels and $K \cap L = \emptyset$, then $K \prec \mathsf{lupl}(K, L)$ and $L \prec \mathsf{lupl}(L, K)$.*

Proof. Assume for contradiction that $K \not\prec \mathsf{lupl}(K, L)$. This means that there is $k \in K$ such that $k \notin \mathsf{lupl}(K, L)$, and there is $l \in L$ such that $tau(k) < tau(l)$ and $l \notin \mathsf{lupl}(K, L)$. The last statement is only valid when $tau(l) < tau(k')$ for some $k' \in K$. By (7), $tau(l) < tau(k')$ and $l \in active(higher(tau(k)))$ implies $tau(k) < tau(k')$, which contradicts the fact that $k, k' \in K$, i.e. k and k' have the same priority. □

One of the consequences of the above lemmas is that for every rebo operator f there is the *top* priority level for f, i.e. a priority level L such that $L \not\prec K$ for every priority level K for f. We denote L as $top(Levels(f))$ and we shall use it in the conditions for the urgency, patience and timelock freeness properties in Section 4.

3.2. Time preserving operators

In this subsection we specify how to extend an arbitrary rebo process operator into a time preserving operator.

Definition 27. Given a rebo operator f and an OSOS operator f', with the sets of rules R and R' and the orderings $<$ and $<'$, respectively, and with no σ actions in R, the operator f' is the *time preserving extension* of f if

1. $R' = R \cup R_\sigma$, where $R_\sigma = \{\sigma_J \mid J \in Levels(f) \ \wedge \ J \text{ is maximal}\}$.
2. $<'$ is the least ordering that contains $< \cup <^{\sigma,\sigma} \cup <^{\tau,\sigma}$ and satisfies conditions (6) and (7), where $<^{\sigma,\sigma}$ and $<^{\tau,\sigma}$ are

   (2.a) $<^{\sigma,\sigma} = \{(r, r') \mid r, r' \in R_\sigma \ \wedge \ active(r) \prec active(r')\}$,

   (2.b) $<^{\tau,\sigma} = \{(tau(k), r) \mid r \in R_\sigma \ \wedge \forall l \in active(r).\ tau(k) < tau(l)\}$.

An operator is *time preserving* if it is the time preserving extension of some rebo operator.

Note that since timed rules of time preserving operators are σ-rules the orderings in (2.a) and (2.b) above can be defined more specifically as

$$<^{\sigma,\sigma} = \{(\sigma_K, \sigma_L) \mid K, L \in Levels(f) \ \wedge \ K \prec L\}$$
$$<^{\tau,\sigma} = \{(tau(k), \sigma_L) \mid \forall l \in L.\ tau(k) < tau(l)\}.$$

However, because Definition 30 below employs corresponding orderings, we keep the more general form of definitions. Moreover, we shall argue that time preserving operators are a special form of time altering operators that we define in the next subsection. This argument will be easier and more clear with the adopted more general form of these definitions.

Example 28. Time preserving extensions of popular process operators are obtained simply by adding the σ-rules for all maximal priority levels for the operators. For example, for the + and parallel composition operators of CCS we add the following σ-rules for the maximal priority level $\{1, 2\}$.

$$\frac{X \xrightarrow{\sigma} X' \quad Y \xrightarrow{\sigma} Y'}{X \parallel Y \xrightarrow{\sigma} X' \parallel Y'}\ \sigma^{\parallel}_{\{1,2\}}, \qquad \frac{X \xrightarrow{\sigma} X' \quad Y \xrightarrow{\sigma} Y'}{X + Y \xrightarrow{\sigma} X' + Y'}\sigma^{+}_{\{1,2\}}.$$

Notice that the rules guarantee that composite processes constructed with + and $\parallel$ can pass time provided that both argument components can pass time. Moreover, the passage

of time does not change the availability of actions. Thus, time synchrony holds: time passes in the composite process if all components with the same priority pass time.

The sequential composition operator ; has two maximal priority levels, namely {1} and {2}, so to achieve its time preserving extension we add two σ-rules:

$$\frac{X \xrightarrow{\sigma} X'}{X; Y \xrightarrow{\sigma} X'; Y}\,\sigma_1, \qquad \frac{Y \xrightarrow{\sigma} Y'}{X; Y \xrightarrow{\sigma} X; Y'}\,\sigma_2.$$

Since $\{2\} \prec \{1\}$ the new ordering on the rules for the time preserving version of ; contains $\sigma_2 < \sigma_1$ by (2.a). As ; is τ-preserving, condition (7) requires $\sigma_2 < r_{a*}$ since $\tau_2 < r_{a*}$. Also, (6) requires $\sigma_2 < \tau_1$ since $\sigma_2 < r_{a*}$. Moreover, we also need $\tau_2 < \sigma_1$ by (2.b), and $r_{*a} < \sigma_1$ by condition (7). The ordering is thus as follows:

$$\begin{aligned} &r_{*a} < r_{a*}, \tau_1, \sigma_1, \\ &\tau^2 < r_{a*}, \tau_1, \sigma_1, \\ &\sigma_2 < r_{a*}, \tau_1, \sigma_1. \end{aligned}$$

The prefixing operators have no active arguments, so their time preserving versions are obtained by adding rules $\sigma_{\emptyset}^{\alpha\cdot} : \alpha.X \xrightarrow{\sigma} \alpha.X$ for each $\alpha \in \mathsf{Act}$. Correspondingly, the time preserving version of the deadlocked process operator $\mathbf{0}$ has one rule $\sigma_{\emptyset}^{\mathbf{0}} : \mathbf{0} \xrightarrow{\sigma} \mathbf{0}$.

Notice that time preserving versions of rebo operators are not necessarily rebo operators themselves. Time preserving extension of + is not a rebo operator because, although it is τ-sensitive, + appears in the target of the conclusion of its σ-rules: this is not allowed by Definition 19.

Not all existing process languages with time have the standard operators extended with time in the above way.

Example 29. In ACP with discrete relative timing and without immediate deadlock [9,10], the passage of time can resolve the choice. If instead of m time units, for $m \in \mathbb{N}$, we use σ, the rules for + become

$$\frac{X \xrightarrow{\sigma} X' \quad Y \xrightarrow{\sigma}}{X + Y \xrightarrow{\sigma} X'}, \qquad \frac{X \xrightarrow{\sigma} \quad Y \xrightarrow{\sigma} Y'}{X + Y \xrightarrow{\sigma} Y'}, \qquad \frac{X \xrightarrow{\sigma} X' \quad Y \xrightarrow{\sigma} Y'}{X + Y \xrightarrow{\sigma} X' + Y'}.$$

We call this form of time synchrony *maximal time synchrony*: time passes in the composite process if all components that can pass time do pass time. Using the OSOS approach the above rules can be equivalently given as

$$\frac{X \xrightarrow{\sigma} X'}{X + Y \xrightarrow{\sigma} X'}\sigma_{\{1\}}, \qquad \frac{Y \xrightarrow{\sigma} Y'}{X + Y \xrightarrow{\sigma} Y'}\sigma_{\{2\}}, \qquad \frac{X \xrightarrow{\sigma} X' \quad Y \xrightarrow{\sigma} Y'}{X + Y \xrightarrow{\sigma} X' + Y'}\sigma_{\{1,2\}},$$

with $\sigma_{\{1\}}, \sigma_{\{2\}} < \sigma_{\{1,2\}}$. This version of + is not time preserving according to our definition but, as will be seen in the next subsection, it is time altering.

3.3. Time altering operators

In this subsection we propose a uniform method for defining time altering operators. We require that time altering operators preserve timed rooted eager bisimulation and the time

determinacy property. These requirements restrict considerably the structure of timed rules and the orderings on rules for such operators. We motivate and derive these restrictions below.

The simplest operator that introduces the passage of time is a *delay* operator $\sigma.X$ [7,27,39,59] defined as follows:

$$\frac{}{\sigma.X \xrightarrow{\sigma} X}.$$

Clearly, $\sigma.$ is rebo. There are many other operators that introduce or alter the passage of time [6,7,26,27,39,42,44,59]. We shall discuss most of these operators in the later sections of this paper.

In general, when defining time altering operators, we will not use rules which change ordinary actions into timed actions, and vice versa. Such rules, if used, can distort the intended difference between timed and ordinary actions, and as a result can fail some desirable timed properties. We illustrate this with an example. Consider operator g which renames every action a of its only argument to σ and leaves other actions intact. The defining rules are as follows:

$$\frac{X \xrightarrow{a} X'}{g(X) \xrightarrow{\sigma} g(X')}, \qquad \frac{X \xrightarrow{\chi} X'}{g(X) \xrightarrow{\chi} g(X')}\ \chi \neq a.$$

Operator g is unsafe because it does not preserve time determinacy. For, $g(a.b + a.c)$ can perform σ actions to several syntactically different processes, for example, $g(b)$ and $g(c)$. Overall, we require that only actions σ can appear in the timed rules of time altering operators.

Next, we propose further restrictions on timed rules and on orderings on such rules so that operators preserve time determinacy. Let an operator h be defined by the following rules.

$$\frac{X \xrightarrow{\sigma} X'}{h(X,Y) \xrightarrow{\sigma} t_1}\ r_1, \qquad \frac{Y \xrightarrow{\sigma} Y'}{h(X,Y) \xrightarrow{\sigma} t_2}\ r_2.$$

Consider $h(p,q)$ with $p \xrightarrow{\sigma} p'$, $q \xrightarrow{\sigma} q'$ and $p' \neq q'$. Clearly, if neither $r_1 < r_2$ nor $r_2 < r_1$, then $h(p,q)$ is not time deterministic if t_1 and t_2 are different open terms. One can think of two restrictions to deal with operators such as h: either demand a timed rule $r_{\{1,2\}}$ for f in addition to the rules above with $r_1, r_2 < r_{\{1,2\}}$, or require $t_1 = t_2$ up to a change of process variables. In our view the second option is not so useful, so we shall only adopt the first. A general formulation of the first alternative will require the usage of the predicate lupl introduced in the previous subsection.

The discussion so far suggests that a given process $f(\boldsymbol{p})$ is time deterministic if, in the presence of general timed rules and non-trivial orderings on rules, there is only one timed rule enabled at $f(\boldsymbol{p})$. This is precisely the motivation for condition (9) in Definition 30: it adjusts the number and the form of timed rules, and the orderings on timed rules for time altering operators.

In general, rules for time altering operators can differ in two ways from rules for time preserving operators. Firstly, time altering operators may use timed rules of general form (including rules with implicit copies) whereas time preserving operators must use σ-rules

which have a very restrictive form. Secondly, the ordering on timed rules for time altering operators are less restrictive than those for time preserving operators.

Definition 30. Given a rebo operator f and an OSOS operator f', with the sets of rules R and R' and the orderings $<$ and $<'$, respectively, and with no σ actions in R, the operator f' is the *time altering extension* of f if

1. $R' = R \cup R_\sigma$, where $R_\sigma = \{r \mid r \text{ is a timed rule } \wedge (active(r) \in Levels(f) \vee active(r) = \emptyset)\}$.
2. $<'$ contains $< \cup <^{\sigma,\sigma} \cup <^{\tau,\sigma}$ and satisfies conditions (6), (7) and (8), where $<^{\sigma,\sigma}$ and $<^{\tau,\sigma}$ are

   (2.a) $<^{\sigma,\sigma} = \{(r, r') \mid r, r' \in R_\sigma \ \wedge \ active(r) \prec active(r')\}$,

   (2.b) $<^{\tau,\sigma} = \{(tau(k), r) \mid r \in R_\sigma \ \wedge \ \forall l \in active(r).\, tau(k) < tau(l)\}$,

and if f' satisfied the following additional condition (9). For all r and r' in R_σ

$$\textbf{if } r \not<' r' \textbf{ and } r' \not<' r \textbf{ then } \exists r'' \in R_\sigma.\, (r, r' <' r'' \textbf{ and } active(r'') = \mathsf{lupl}(active(r), active(r'))). \tag{9}$$

An operator is *time altering* if it is the time altering extension of some rebo operator.

Condition (9) has been motivated by the example of the operator h above. It is needed because there are timed operators which do not satisfy it, and thus fail time determinacy. The examples are the timed version of the ACP + [21] and the operator $\oplus$ in [34]. The timed rules for the first operator are very similar to the rules for the operator h above, of course with $X + Y$ replacing $h(X, Y)$, and with X' and Y' replacing t_1 and t_2, respectively. The two timed rules are not ordered. However, there is no third time rule, with the active arguments X and Y, which is above the first two rules as required by (9).

Note that time preserving operators are a special form of time altering operators in the following sense:

- Set R_σ in Definition 30 contains all the maximal σ-rules for f and all its priority levels, and no other timed rules.
- The ordering $<'$ is the least ordering that contains $< \cup <^{\sigma,\sigma} \cup <^{\tau,\sigma}$ and satisfies the conditions stated in part 2 of Definition 27.

Moreover, since we have limited the form of timed rules to σ-rules, thus removing the (remote in practice) possibility of implicit copies in the timed rules, the above ordering $<'$ satisfies additionally condition (8) as required in part 2 of Definition 30. Also, it is easy to verify that the above rules and the ordering satisfy condition (9). Hence, time preserving operators are also time altering. Clearly, the converse is not valid.

Example 31. For all timed operators discussed in [6,7,26,27,39,42,44,59] we can easily find their time altering versions. This is with a small proviso: the operators' definitions are slightly changed by including the associated τ-rules or silent choice rules, and by using σ in transitions instead of t units of time.

As an example, we give an alternative definition of the delay operator "$\lfloor\ \rfloor(\ )$" of *Temporal Process Language* (TPL) of Hennessy and Regan [27]. The original rules for the operator are as follows:

$$\frac{X \xrightarrow{a} X'}{\lfloor X \rfloor(Y) \xrightarrow{a} X'}, \qquad \frac{X \xrightarrow{\tau} X'}{\lfloor X \rfloor(Y) \xrightarrow{\tau} X'}, \qquad \frac{X \xrightarrow{\tau}}{\lfloor X \rfloor(Y) \xrightarrow{\sigma} Y}.$$

Here, we do not need to add any silent rules and we can reuse σ. We redefine $\lfloor\ \rfloor(\ )$ with ordered rules instead of rules with negative premises:

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

and the ordering is $\sigma_\emptyset <' \tau^1$. We can easily check that $\lfloor\ \rfloor(\ )$ is a τ-sensitive operator. The operator is also time altering: $\lfloor\ \rfloor(\ )$ is obtained by extending the operator defined by the first two rules above. Note, the first two rules are unordered. We add rule $\sigma_\emptyset$, and extend the ordering. Rule $\sigma_\emptyset$ is a timed rule with no active arguments. Note that it is not a σ-rule: firstly, since the first argument of $\lfloor\ \rfloor(\ )$ is active the rule should have the premise $X \xrightarrow{\sigma} X'$, and, secondly, the operator $\lfloor\ \rfloor(\ )$ does not appear in its target.

As for the ordering $<'$ required by Definition 30, we have the following. The original ordering $<$ is empty. The components $<^{\sigma,\sigma}$ and $<^{\tau,\sigma}$ are also empty since there is only one timed rule and $\lfloor\ \rfloor(\ )$ has only one active argument. The ordering $\sigma_\emptyset <' \tau^1$ is the extra component of $<'$ permitted by the "contains" phrase in part 2 of Definition 30. Finally, we easily check that both the rules and the ordering satisfy condition (9).

Notice that $\lfloor\ \rfloor(\ )$ does not preserve eager bisimulation $\sqsubseteq\!\!\sim$. But, since $\lfloor\ \rfloor(\ )$ is a rebo operator it preserves $\sqsubseteq\!\!\sim_r$.

There are only few timed versions of traditional process operators in the concurrency literature which are neither time preserving nor time altering. The examples are the + in [21] and $\oplus$ in [34]; and the resulting LTSs for these time process languages are *not* time deterministic.

3.4. Timed process languages

In this subsection we define a general class of timed process languages and show that member languages satisfy time determinacy and preserve a timed version of rooted eager bisimulation.

We begin with the definition of the timed rooted eager bisimulation relation.

Definition 32. Given $(\mathcal{P}, \mathsf{Act}_\sigma, \rightarrow)$, a relation $R \subseteq \mathcal{P} \times \mathcal{P}$ is a *timed rooted eager bisimulation* if, for all $(p, q) \in R$, the following properties hold.

$(R_\sigma.a1)$ $\forall\alpha.\ (p \xrightarrow{\alpha} p' \text{ implies } \exists q', q''.\ (q \stackrel{\tau}{\Rightarrow} q' \xrightarrow{\alpha} q'' \text{ and } p' \sqsubseteq\!\!\sim q''))$

$(R_\sigma.a2)$ $p \xrightarrow{\sigma} p' \text{ implies } \exists q'.\ (q \xrightarrow{\sigma} q' \text{ and } p' R q')$

$(R_\sigma.c1)$ $p\Downarrow \text{ implies } \forall\alpha.\ (q \xrightarrow{\alpha} q' \text{ implies } \exists p', p''.\ (p \stackrel{\tau}{\Rightarrow} p' \xrightarrow{\alpha} p'' \text{ and } p'' \sqsubseteq\!\!\sim q'))$

$(R_\sigma.c2)$ $p\Downarrow \text{ implies } (q \xrightarrow{\sigma} q' \text{ implies } \exists p'.\ (p \xrightarrow{\sigma} p' \text{ and } p' R q'))$

We write $p \sqsubseteq\!\!\sim_{tr} q$ if there exists a timed rooted eager bisimulation R such that pRq.

If the LTS we are considering has no actions σ, then $\sqsubseteq\!\!\sim_{tr}$ coincides with $\sqsubseteq\!\!\sim_r$. Otherwise, $\sqsubseteq\!\!\sim_{tr}$ is a proper subset of $\sqsubseteq\!\!\sim_r$: although $\sigma.a \sqsubseteq\!\!\sim_r \sigma.\tau.a$ but $\sigma.a \not\sqsubseteq\!\!\sim_{tr} \sigma.\tau.a$. The root condition not only applies to the initial states of the two related processes but also to any states that are reachable from the initial states by performing time transitions.

Definition 33. A process language is called timed rebo, or simply *timed process language*, if its operators can be partitioned into time preserving operators and time altering operators,

and the targets of all timed rules for time altering operators do not contain τ-sensitive operators.

A typical timed process language consists of two groups of operators. The first group contains time preserving versions of operators of popular process languages, for example CCS or CSP or ACP, provided that the operators are rebo. One may also employ new task specific operators as long as they are the time preserving versions of rebo definable operators. All these operators pass time in the deterministic fashion, and observe a form of time synchrony. The passage of time is introduced by operators from the second group, namely the time altering operators. Only together with operators from the first group, they create a useful process language with discrete relative time.

The difference between rebo and timed rebo operators is that timed rebo operators may have rules with targets containing τ-sensitive operators—such rules are forbidden for rebo operators. These rules are the σ-rules for time preserving operators which are also τ-sensitive as, for example, the CCS time preserving +. Thus, in general, timed rebo operators do not preserve rooted eager bisimulation. Consider the mentioned +. We have $\sigma.a \sqsubseteq_r \sigma.\tau.a$ but $p = \sigma.a + \sigma.b \not\sqsubseteq_r \sigma.\tau.a + \sigma.b = q$ since $p \xrightarrow{\sigma} a + b$ and $q \xrightarrow{\sigma} \tau.a + b$, and clearly $a + b \not\sqsubseteq \tau.a + b$. However, because of the form of σ-rules for time preserving operators (Definition 27) and a suitably defined timed rooted eager bisimulation, we have the promised congruence result. However, first, we prove that timed process languages preserve the time determinacy property.

Theorem 34. *Let L be any timed process language and T be its LTS. Then, T satisfies the time determinacy property.*

Proof. We show, by induction on the operator depth of process terms, that given a process p over T, $p \xrightarrow{\sigma} p'$ and $p \xrightarrow{\sigma} p''$ imply $p' = p''$. In fact, it is sufficient to show that there cannot be two different timed rules enabled at p.

If p is a constant, say f, then its SOS rules have no premises and thus no active arguments. When f is time preserving, it has precisely one timed rule: $\sigma^f_\emptyset : f \xrightarrow{\sigma} f$; and time determinacy holds. When f is time altering, it may have several timed rules, possibly ordered, according to Definition 30. Assume for contradiction that $f \xrightarrow{\sigma} f'$ and $f \xrightarrow{\sigma} f''$ are derivable by different timed rules r and r', respectively, with $active(r) = active(r') = \emptyset$. This means that r and r' are enabled at f, and $r \not< r'$ and $r' \not< r$. Now, we make use of condition (9). Since $\mathsf{lupl}(\emptyset, \emptyset) = \emptyset$, the condition requires a timed rule r'' for f with no active arguments which is above r and r'. Hence, r'' itself or a timed rule higher than r'' is enabled at $f(\boldsymbol{u})$: contradiction. Hence, f has a unique σ-derivative.

Let $p = f(\boldsymbol{u})$ where f is an n-ary operator. By the inductive hypothesis, each component u_i of $\boldsymbol{u}$ which is able to pass time has a unique σ-derivative, say u'_i. We show that there cannot be two timed rules enabled at $f(\boldsymbol{u})$. Suppose for contradiction that $f(\boldsymbol{u}) \xrightarrow{\sigma} p'$ is derived by a timed rule r and $f(\boldsymbol{u}) \xrightarrow{\sigma} p''$ by a timed rule r', and r and r' are different. Since both r, r' are enabled at $f(\boldsymbol{u})$, we deduce $r \not< r'$ and $r' \not< r$. We consider active arguments of r and r'.

Assume without loss of generality that $active(r) = \emptyset$. If f is time preserving, then, by Definition 27, r is the only timed rule for f with $active(r) = \emptyset$. Hence, $r = r'$: contradiction. If f is time altering, then either $active(r') \neq emptyset$ or $active(r') = \emptyset$.

In the first case $active(r) \subset active(r')$, hence $active(r) \prec active(r')$. By condition (2.a) of Definition 30 we have $r < r'$: contradiction. In the second case we use condition (9). Since $\mathsf{lupl}(\emptyset, \emptyset) = \emptyset$, the condition requires a timed rule r'' with no active arguments which is above r and r'. As r'' has no arguments, r'' itself or a timed rule higher than r'' is enabled at $f(\boldsymbol{u})$: contradiction.

Next, let $active(r) \neq \emptyset$ and $active(r') \neq \emptyset$. Clearly, $active(r)$ and $active(r')$ are priority levels. By Lemma 25, $\mathsf{lupl}(active(r), active(r'))$ is also a priority level for f. Hence, by Lemma 26, $active(r) \prec \mathsf{lupl}(active(r), active(r'))$ and $active(r') \prec \mathsf{lupl}(active(r), active(r'))$.

If f is time preserving, then it has the σ-rule, say r'', for the priority level $\mathsf{lupl}(active(r), active(r'))$. If f is time altering, then since $r \not< r'$ and $r' \not< r$, by condition (9), there must exist a timed rule, say r'', with active arguments $\mathsf{lupl}(active(r), active(r'))$ which is above r and r'. It is clear that in both cases r'' is applicable to $f(\boldsymbol{u})$ to derive a σ transition. Hence, r'' itself or a timed rule higher than r'' is enabled at $f(\boldsymbol{u})$: contradiction.

Thus, we have shown that there is at most one timed rule enabled at $f(\boldsymbol{u})$. By the inductive hypothesis, the subterms u_i are time deterministic. Hence, $f(\boldsymbol{u})$ has a unique σ-derivative. □

Next, we have the congruence theorem for the timed process languages. The proof is similar to that of Theorem 22 and is given in Appendix B.

Theorem 35. *All timed process languages preserve timed rooted eager bisimulation.*

4. Properties of timed process languages

Apart from time determinacy, there are several other important timed properties which are often discussed in the concurrency literature [26,29,38,44,59]. Fig. 2 lists the properties that we shall consider in this paper. For brevity, we leave out in Fig. 2 the outermost universal quantifies binding processes p and action a where appropriate. There are other timed properties which are not considered here. Firstly, there is the *time additivity property*: if $p \xrightarrow{s} p'$ and $p' \xrightarrow{t} p''$, for some p' and p'', then $p \xrightarrow{s+t} p''$, and vice versa, where s and t indicate the passage of s and t time units respectively. Since we consider the passage of

time determinacy	**if** $p \xrightarrow{\sigma} p'$ **and** $p \xrightarrow{\sigma} p''$ **then** $p' = p''$
timelock freeness	$p \xrightarrow{\sigma}$
weak timelock freeness	**if** $p\Downarrow$ **then** $p \stackrel{\sigma}{\Rightarrow}$
maximal progress	**if** $p \xrightarrow{\tau}$ **then** $p \stackrel{\sigma}{\not\rightarrow}$
patience	**if** $p \stackrel{\tau}{\not\rightarrow}$ **then** $p \xrightarrow{\sigma}$
constancy of offers	**if** $p \xrightarrow{\sigma} p'$ **then** $(p \xrightarrow{a}$ **iff** $p' \xrightarrow{a})$
time persistence	**if** $p \xrightarrow{\sigma} p'$ **and** $p \xrightarrow{a}$ **then** $p' \xrightarrow{a}$
urgency	$q \stackrel{\tau}{\not\rightarrow}$ **and** $q \stackrel{\sigma}{\not\rightarrow}$, for some q

Fig. 2. Timed properties.

time in terms of time units, the additivity is trivially satisfied. Secondly, these are *finite* and *bounded variability* properties [38,42].

There are dependencies between some of the properties in Fig. 2.

Lemma 36. *The negation of the urgency property is equivalent to the patience property, and the patience property implies the weak timelock freeness property. Constancy of offers implies time persistence.*

In the following subsections and later on we shall often say that an operator satisfies a timed property. In most cases this means that, given an n-ary operator f and a property P, if a vector of processes $\boldsymbol{p}$ satisfies P, then $f(\boldsymbol{p})$ also satisfies P. If there is a difference from this definition, we shall state it clearly.

4.1. Maximal progress

A process satisfies maximal progress if whenever it is able to perform a τ, then it must not be able to pass time. Recall, that an n-ary operator f satisfies maximal progress if, for some n-ary vector of processes $\boldsymbol{p}$ that satisfy maximal progress, $f(\boldsymbol{p}) \xrightarrow{\tau}$ implies $f(\boldsymbol{p}) \overset{\sigma}{\not\to}$.

Let L be any timed process language and T be its LTS. We claim that if all operators f in L satisfy the following condition, then T satisfies the maximal progress property.

For all rules r and r' for f, and all $J \in \mathit{Levels}(f)$

$$\textbf{if } \mathit{act}(r) = \sigma \textbf{ and } \mathit{act}(r') = \tau \textbf{ and } \mathit{active}(r) \cup \mathit{active}(r') \subseteq J \textbf{ then } r < r'.$$

Informally, the condition says that for each operator all timed rules must be below all the rules with the action τ. Although quite intuitive, the condition is too strong and can be weakened. We illustrate this by analysing two operators, namely the parallel composition of CCS and the *mixed choice* operator "$\triangleright$" from [48], in the setting of a time preserving extension of CCS with the delay operator "Δ" defined below. We begin with the mixed choice. Informally, $p \triangleright q$ is a process which behaves like p or like q, but the choice of q can only be made internally and independently of the environment. The rules for $\triangleright$ are

$$\frac{X \xrightarrow{a} X'}{X \triangleright Y \xrightarrow{a} X'}, \quad \frac{X \xrightarrow{\tau} X'}{X \triangleright Y \xrightarrow{\tau} X' \triangleright Y}\ \tau_1, \quad \frac{}{X \triangleright Y \xrightarrow{\tau} Y}\ r_\emptyset, \quad \frac{X \xrightarrow{\sigma} X'}{X \triangleright Y \xrightarrow{\sigma} X' \triangleright Y}\ \sigma_1.$$

The operator $\triangleright$ has the single priority level $\{1\}$, hence σ-rule σ_1. It has the empty ordering on its rules. For the maximal progress property to be valid, we need to order the above rules (as well as rules for other operators of the process language) according to the condition. The required ordering is $\sigma_1 < \tau_1$ and $\sigma_1 < r_\emptyset$. When p and q satisfy maximal progress, the ordering $\sigma_1 < \tau_1$ is spurious. This is because if $p \xrightarrow{\tau}$, then $p \overset{\sigma}{\not\to}$ by the maximal progress property. Hence, τ_1 is applicable and enabled, but σ_1 is not applicable. Consequently, $p \triangleright q \xrightarrow{\tau}$ and $p \triangleright q \overset{\sigma}{\not\to}$. On the other hand, $\sigma_1 < r_\emptyset$ is essential to guarantee maximal progress. Note that $\triangleright$ is time altering.

The situation is somewhat different with a timed version of parallel composition operator of CCS. Its rules are

$$\frac{X \xrightarrow{\alpha} X'}{X \parallel Y \xrightarrow{\alpha} X' \parallel Y}\ r_{\alpha *}, \quad \frac{Y \xrightarrow{\alpha} Y'}{X \parallel Y \xrightarrow{\alpha} X \parallel Y'}\ r_{* \alpha},$$

$$\frac{X \xrightarrow{a} X' \quad Y \xrightarrow{\bar{a}} Y'}{X \parallel Y \xrightarrow{\tau} X' \parallel Y'}\, r_{a\bar{a}}, \qquad \frac{X \xrightarrow{\tau} X'}{X \parallel Y \xrightarrow{\tau} X' \parallel Y}\, \tau_1, \qquad \frac{Y \xrightarrow{\tau} Y'}{X \parallel Y \xrightarrow{\tau} X \parallel Y'}\, \tau_2,$$

$$\frac{X \xrightarrow{\sigma} X' \quad Y \xrightarrow{\sigma} Y'}{X \parallel Y \xrightarrow{\sigma} X' \parallel Y'}\, \sigma_{\{1,2\}}, \qquad \frac{X \xrightarrow{\sigma} X'}{X \parallel Y \xrightarrow{\sigma} X' \parallel Y}\, \sigma_1, \qquad \frac{Y \xrightarrow{\sigma} Y'}{X \parallel Y \xrightarrow{\sigma} X \parallel Y'}\, \sigma_2$$

and the ordering required by (2.a) of Definition 27 is $\sigma_1, \sigma_2 < \sigma_{\{1,2\}}$. According to the condition, we need to extend the ordering to include $\sigma_{\{1,2\}} < r_{a\bar{a}}, \tau_1, \tau_2$ and $\sigma_1, \sigma_2 < r_{a\bar{a}}, \tau_1, \tau_2$. Note that this would make the operator time altering. The constraint $\sigma_{\{1,2\}} < r_{a\bar{a}}$ is mandatory. For the same reasons as for $\triangleright$ above, we do not need the ordering to satisfy $\sigma_{\{1,2\}}, \sigma_1, \sigma_2 < \tau_1, \tau_2$ when p and q satisfy maximal progress. However, unlike for $\triangleright$, the ordering defined so far satisfies these constraints anyway, whether or not they are imposed by the condition: Since $\sigma_1, \sigma_2 < \sigma_{\{1,2\}}$, we obtain $\sigma_1, \sigma_2 < \tau_1, \tau_2$ by condition (6). Correspondingly, from $\sigma_{\{1,2\}} < r_{a\bar{a}}$ we deduce $\sigma_{\{1,2\}} < \tau_1, \tau_2$.

In general, the full strength of the above condition is required for time altering operators. Consider the rules for Δ:

$$\frac{X \xrightarrow{a} X'}{\Delta X \xrightarrow{a} \Delta X'}\, r_a, \qquad \frac{X \xrightarrow{\tau} X'}{\Delta X \xrightarrow{\tau} \Delta X'}\, \tau_1, \qquad \frac{}{\Delta X \xrightarrow{\sigma} X}\, r_\sigma.$$

Without the ordering $r_\sigma < \tau_1$, we have $\Delta p \xrightarrow{\tau}$ and $\Delta p \xrightarrow{\sigma}$ for p that satisfies maximal progress. This is because rule r_σ is always enabled. The problem is solved by ordering τ_1 and r_σ according to the condition: $r_\sigma < \tau_1$.

As a result of this discussion, maximal progress holds under weaker conditions:

Theorem 37. *Let L be any timed process language and T be its LTS. If every time preserving operator f of L satisfies* (10) *below and every time altering operator g of L satisfies* (11) *below, then T satisfies the maximal progress property.*

For all rules r and r' for f, and all priority levels J for f,

$$\textbf{if } act(r) = \sigma \textbf{ and } act(r') = \tau \textbf{ and not } silent(r') \textbf{ and } active(r) \cup active(r') \subseteq J \textbf{ then } r < r'. \tag{10}$$

For all rules r and r' for g, and all priority levels J for g,

$$\textbf{if } act(r) = \sigma \textbf{ and } act(r') = \tau \textbf{ and } active(r) \cup active(r') \subseteq J \textbf{ then } r < r'. \tag{11}$$

Proof. We show, by induction on the operator depth of process terms, that given a process p over T, if $p \xrightarrow{\tau}$, then $p \xrightarrow{\sigma}$. Let f be an operator of L and let $J \in Levels(f)$.

If p is a constant, say f, then $Levels(f) = \{\emptyset\}$. Since all rules for f have no premises, they are all applicable. Consider the rules for f. If there are no timed rules or rules with the action τ, then vacuously maximal progress holds for f. Otherwise, let R_τ be the set of rules for f with the action τ, and let R_σ be the set of timed rules for f. Condition (10) and (11) require that every rule $r \in R_\sigma$ is below every rule $r' \in R_\tau$. Since all rules in R_τ are applicable, we infer that either $f \xrightarrow{\tau}$, or $f \xrightarrow{\tau}$ and so no rule in R_σ is ever enabled. In both cases maximal progress holds.

Suppose that $p = f(\boldsymbol{p})$ and $f(\boldsymbol{p}) \xrightarrow{\tau}$ is derivable by a rule with active arguments in J. By the inductive hypothesis, all active arguments p_j, for $j \in J$, satisfy maximal progress. There are three cases.

1. f is time preserving and $f(\boldsymbol{p}) \xrightarrow{\tau}$ is derived by a silent rule $tau(i)$, where $i \in J$. By the inductive hypothesis, σ-rule σ_i is not applicable to $f(\boldsymbol{p})$ as $p_i \overset{\sigma}{\not\to}$. It may happen that a σ-rule σ_I, for $I \subseteq J \setminus \{i\}$, is applicable to $f(\boldsymbol{p})$. But, since $\sigma_I < \sigma_J$ we obtain $\sigma_I < tau(j)$, for all $j \in J$, by condition (6). This includes $\sigma_I < tau(i)$, so σ_I is not enabled at $f(\boldsymbol{p})$. Hence, $f(\boldsymbol{p}) \overset{\sigma}{\not\to}$. Notice, that this is the reason why a weaker condition (10) can be used for time preserving operators.
2. f is time altering and $f(\boldsymbol{p}) \xrightarrow{\tau}$ is derived by a silent rule $tau(i)$, where $i \in J$. By the inductive hypothesis, any timed rule with active argument i is not applicable to $f(\boldsymbol{p})$. There may be timed rules with active arguments in $J \setminus \{i\}$ applicable to $f(\boldsymbol{p})$, but condition (11) demands that they all are below $tau(i)$, thus not enabled. Among these timed rules, there may be a rule with empty premises, say $r_\emptyset$, as allowed by Definition 30. If there are no timed rules higher than $r_\emptyset$, then (6) is not sufficient to deduce $r_\emptyset < tau(i)$ (see Δ operator above). Thus, (11) is required to make this deduction.
3. $f(\boldsymbol{p}) \xrightarrow{\tau}$ is derived by a non-silent rule r_I, with active arguments $I \subseteq J$. Conditions (10) and (11) require $r < r_I$ for every timed rule r for f with $active(r) \subseteq J$. Hence, since r_I is enabled at $f(\boldsymbol{p})$, no rule r is enabled at $f(\boldsymbol{p})$, so $f(\boldsymbol{p}) \overset{\sigma}{\not\to}$. Note that I may be empty. □

Example 38. Consider a timed process language which is obtained by putting together $\lfloor\ \rfloor(\ )$ from Example 31 and the timed extensions of CCS operators. We easily check that $\lfloor\ \rfloor(\ )$ satisfies (11) as it is time altering. We order the rules for the prefixing with τ as follows: $\tau.X \xrightarrow{\sigma} \tau.X < \tau.X \xrightarrow{\tau} X$. Since $\tau.X \xrightarrow{\tau} X$ is always applicable it follows that the σ-rule is never applicable for processes of the form $\tau.p$. As for the parallel composition operator, in order to satisfy (10) or (11) we need to put its σ-rules below at least its synchronisation rule. This makes the parallel (and the prefixing with τ) time altering operators as there is a non-empty ordering between their timed and action rules. Other operators of CCS, namely the choice, renaming and restriction, are extended in the time preserving fashion. Hence, there is no ordering between their silent rules and timed rules. Clearly, this satisfies (10). Thus, the resulting process language, henceforth called *TL*, satisfies maximal progress. *TL* is also time deterministic by Theorem 34.

ET-LOTOS [30,31] does not satisfy the maximal progress property since the process $\mathbf{i}\{1\}; P$, whose syntax is explained in Section 5.5, can both idle for one time unit and perform τ immediately. However, ET-LOTOS satisfies a milder property called the *maximal progress on hidden actions*. This property states that any hidden action must occur as soon as possible and cannot be delayed. We discuss this property in more detail in Section 5.5.

4.2. Urgency

A process is urgent if it is able to perform a visible action without any delay or it is completely deadlocked. A transition system is urgent if it has an urgent process. Typically, the passage of time in process languages is introduced by time altering operators, and it is

these operators that may cause urgency. So, we say that an n-ary operator f is *urgent* if, for some n-ary vector of non-urgent processes $\boldsymbol{p}$, we have $f(\boldsymbol{p}) \stackrel{\tau}{\not\rightarrow}$ and $f(\boldsymbol{p}) \stackrel{\sigma}{\not\rightarrow}$.

In the remainder of this section we shall use a new notation which we now motivate. Consider a simple timed process language with time preserving **0**, prefixing and the sequential composition operator ; from Example 28. Operator ; has two priority levels {1} and {2}, with {1} being the top priority level. All the rules for the second argument are below the rules for the first argument, including σ_1. As a result, process q is never activated in $p;q$ for all p and q over the language. More formally, no rule for the second argument for ; is enabled at $p;q$ for any patient p (i.e. $p \stackrel{\tau}{\rightarrow}$ or $p \stackrel{\sigma}{\rightarrow}$). This is because either σ_1 or the silent rule τ_1 are always enabled at $p;q$ for any patient p. In order to construct an urgent version of ; we need to disable σ_1 and τ_1. However, we do not need to disable σ_2 or τ_2. More generally, consider an operator f with several priority levels and several timed rules for each level. In order to make f urgent it is sufficient to disable the "maximal" timed rule among the timed rules for the top priority level and to disable its non-silent rules with action τ. Since patience is the negation of urgency, in order to guarantee that a timed process language is patient we must ensure that no operator has its maximal timed rule for the top priority level disabled and its rules with τ disabled. Hence, the new notation: A timed rule r for f is *maximal* for a priority level J if there is no other timed rule r' for f with $active(r) \subseteq active(r') \subseteq J$ and $r < r'$. Because of the form of timed rules for time preserving operators, and because of condition (9) for time altering operators, if timed rules exist for f, then there is the unique maximal timed rule for every priority level for f. We present two intuitive conditions that guarantee urgency.

For all rules r for f, there exists a rule r' for f such that

$$\begin{array}{l}\textbf{if } act(r) = \tau \textbf{ and } \textbf{not } silent(r) \textbf{ and } active(r) \subseteq top(Levels(f)) \\ \quad \textbf{then } \textbf{not } silent(r') \textbf{ and } r < r'. \end{array} \tag{12}$$

For all rules r for f, there exists a rule r' for f such that

$$\begin{array}{l}\textbf{if } act(r) = \sigma \textbf{ and } active(r) \subseteq top(Levels(f)) \\ \quad \textbf{then } \textbf{not } silent(r') \textbf{ and } r < r'. \end{array} \tag{13}$$

The first condition says that all rules with action τ, which are not silent rules, with active arguments among the top priority level are below a certain non-silent rule. Clearly, the active arguments of that rule are members of the top priority level.

The second condition should be considered in conjunction with condition (9) for time altering operators. Since in Theorem 39 we only consider the top priority level J, (9) implies that every timed rule for J, except for the maximal timed rule, has a timed rules above it. Additionally, (13) requires that the maximal timed rule for J has a non-silent rule above it. We easily deduce that the active arguments of that non-silent rule belong to J.

Theorem 39. *Let L be any timed process language that contains time preserving versions of CCS* **0**, *prefixing and* $+$, *and let T be the LTS generated by L. If there is a time altering operator f in L, which satisfies conditions* (12) *and* (13), *then T satisfies the urgency property.*

Urgent prefixing $a{:}X$ [34,39] is a time altering urgent operator. Its defining rule is $a{:}X \stackrel{a}{\rightarrow} X$. It has a single priority level $\emptyset$, and has no σ-rules and no τ-rules, thus satisfying

(12) and (13). A completely deadlocked process operator called *timelock* [34,39], which has no defining rules, is also a time altering urgent operator. Such an operator is needed in timed process languages with sequential composition as discussed in Example 28. Several forms of urgent prefixing and deadlocked operators appear in the timed versions of ACP where sequential composition is defined somewhat differently. They are described in detail in Section 5.6. The notion of urgency also appears in [17].

Proof. Proof of Theorem 39. Let f be a time altering n-ary operator of L, and let f satisfy (12) and (13). Assume that J is the top priority level of f. We shall construct two urgent processes depending on the OSOS definition of f.

If $J = \emptyset$, then all rules for f have no premises and are applicable to $f(\mathbf{0}, \ldots, \mathbf{0})$. We can show in the corresponding fashion to the argument below for the case $J \neq \emptyset$, that $f(\mathbf{0}, \ldots, \mathbf{0})$ is urgent.

Let $J \neq \emptyset$. If there are no timed rules and no rules with action τ for f, then $f(\mathbf{0}, \ldots, \mathbf{0})$ is urgent. Otherwise, let the maximal timed rule for f and J be r_σ, and let R_τ and R_σ be the sets of non-silent rules for f with action τ and timed rules for f, respectively. Condition (12) guarantees that there is a set $R^{>\tau}$ of non-silent rules for f such that each rule in R_τ is below a rule in $R^{>\tau}$. Note, $R^{>\tau}$ may contain timed rules as well as rules with action τ, and it must contain at least one rule with a non-τ action. Condition (13) requires that there is a set $R^{>\sigma}$ of non-silent rules for f such that each rule in R_σ is below a rule in $R^{>\sigma}$. Note, $R^{>\sigma}$ may contain timed rules as well as rules with action τ, and it must contain at least one non-timed rule. Let K be the set of all active arguments of the rules in $R^{>\tau} \cup R^{>\sigma}$; clearly $K \subseteq J$.

Now, we are ready to construct the urgent process $f(\boldsymbol{p})$. If $i \notin K$ or all rules in $R^{>\tau} \cup R^{>\sigma}$ have premises for the ith argument only with the action σ, i.e. of the form $X_i \xrightarrow{\sigma} X_i'$, then $p_i = \mathbf{0}$. Otherwise, $p_i = \sum_{i \in K} a_{il}.\mathbf{0}$, where $a_{il}.\mathbf{0}$ is a summand if and only if there is a rule $r \in R^{>\tau} \cup R^{>\sigma}$ with the premise $X_i \xrightarrow{a_{ik}} X_{ik}$ (up to a change of variable names) and $a_{il} = a_{ik}$. We easily verify that each p_i is non-urgent: $p_i \xrightarrow{\sigma}$ and $p_i \stackrel{\tau}{\not\rightarrow}$. Due to the form of processes p_i, all non-timed rules in $R^{>\tau} \cup R^{>\sigma}$ are applicable. Also, since $p_i \xrightarrow{\sigma}$ for all i, all timed rules for f and J are applicable. Hence, the rules in $R^{>\tau} \cup R^{>\sigma}$ disable all the rules in $R_\tau \cup R_\sigma$. Note that some rules from $R_\tau \cup R_\sigma$ may appear in $R^{>\tau} \cup R^{>\sigma}$. For example, r_σ may be disabled by some non-timed rule in $R^{>\sigma}$, and all timed rules in R_σ except for r_σ are disabled by r_σ itself. Overall, none of the rules in $R_\tau \cup R_\sigma$ is enabled. If there are rules higher than those in $R^{>\tau} \cup R^{>\sigma}$ that are applicable to $f(\boldsymbol{p})$, they have visible actions by the two urgency conditions. Note that there may be silent rules that have the same or higher priority than the rules in $R_\tau \cup R_\sigma$, but they are not applicable to $f(\boldsymbol{p})$ as all subprocesses p_i are stable. Thus, $f(\boldsymbol{p}) \stackrel{\tau}{\not\rightarrow}$ and $f(\boldsymbol{p}) \stackrel{\sigma}{\not\rightarrow}$ as required. □

4.3. Patience and weak timelock freeness

In the setting of timed process languages as presented in this paper, and influenced by the dependencies in Lemma 36, we have the following intuitive conditions on the operators to guarantee the patience property.

For r_σ, the maximal timed rule for f and J, and for all rules r for f

$$\textbf{if} \quad r_\sigma < r \quad \textbf{then} \quad act(r) = \tau. \tag{14}$$

For r_σ, the maximal timed rule for f and J, and for all rules r and r' for f

if $r_\sigma < r$ **and** $r < r'$ **then** $silent(r')$. (15)

The conditions allow only rules with the action τ, whether silent or non-silent rules, to be above the maximal timed rule for the top priority level for a typical timed operator. Moreover, if such rules with the action τ exist, then only silent rules can be above them. Informally, this means that, given process $f(\boldsymbol{p})$, either the maximal timed rule for f and its top priority level or a rule with action τ is enabled at $f(\boldsymbol{p})$, thus giving $f(\boldsymbol{p}) \stackrel{\sigma}{\rightarrow}$ or $f(\boldsymbol{p}) \stackrel{\tau}{\rightarrow}$. The reason why it is sufficient to consider only the top priority levels of timed operators is that, once (14) and (15) are satisfied by all operators, the behaviour of timed processes is determined alone by the OSOS rules for the top priority levels of the operators.

Lemma 36 tells us that if a timed process language satisfies patience, then it also satisfies the weak timelock freeness property. It is easy to verify that time preserving operators satisfy (14) and (15) as no rule is above the maximal σ-rule. Hence, the patience property of timed process languages depends on time altering operators satisfying the conditions.

Theorem 40. *Let L be any timed process language and T be the LTS generated by L. If each time altering operator f in L, with $J = top(Levels(f))$, has a non-empty set of timed rules which satisfy* (14) *and* (15), *then the patience and weak timelock freeness properties hold in T.*

Proof. We show, by induction on the operator depth of process terms, that given a process p over T, $p \stackrel{\tau}{\rightarrow}$ or $p \stackrel{\sigma}{\rightarrow}$, which is equivalent to if $p \stackrel{\tau}{\not\rightarrow}$, then $p \stackrel{\sigma}{\rightarrow}$. Assume that f is the outermost operator of p, J is the top priority level for f, and the set of timed rules for f is not empty. Let all time altering operators f satisfy (14) and (15). Note that time preserving operators already satisfy both of the conditions.

If p is a constant, then $p = f$ for some f. Also, $Levels(f) = \{\emptyset\}$ and r_σ is $f \stackrel{\sigma}{\rightarrow} f'$ for some f' of L. All rules for f are applicable since they have no premises. As f satisfies (14), there are no rules with visible actions above r_σ. Note, since rules for f have no premises, silent rules for f that could be above r_σ by (14) do not exist. Hence, due to (14), the only type of rules that could be above r_σ are the non-silent rules with action τ. Let R_τ be the set of such rules. As there are no silent rules for f, (15) requires that no rule is above any rule in R_τ. If $R_\tau = \emptyset$, then r_σ is enabled and $f \stackrel{\sigma}{\rightarrow}$. Otherwise, every rule in R_τ is enabled, so $f \stackrel{\tau}{\rightarrow}$. In both cases f is patient.

Suppose $p = f(\boldsymbol{p})$. By the inductive hypothesis all arguments p_j, for $j \in J$, satisfy patience, namely $p_j \stackrel{\tau}{\rightarrow}$ or $p_j \stackrel{\sigma}{\rightarrow}$. (14) permits only two types of rules above r_σ. Let R_τ be the set of non-silent rules with action τ which are above r_σ, and let R^τ be the set of silent rules above r_σ. Moreover, (15) permits only silent rules above rules in R_τ; assume that R_{τ^τ} is the set of such rules. Note that some rules with visible actions may be above some non-silent rules with action τ which are not members of R_τ. No rules can be above rules in R^τ and R_{τ^τ}, otherwise by (6) we obtain $tau(i) < tau(j)$, for some $i, j \in J$, contradicting the assumption that J is a priority level. Similarly, we can show that silent rules for J which are not in R^τ are not below r_σ.

Consider subprocesses p_j for $j \in J$. By the inductive hypothesis, either $p_j \stackrel{\tau}{\rightarrow}$ for some $j \in J$, or $p_j \stackrel{\tau}{\not\rightarrow}$ and $p_j \stackrel{\sigma}{\rightarrow}$ for all $j \in J$. In the first case $f(\boldsymbol{p}) \stackrel{\tau}{\rightarrow}$ by the silent rule for j. In the second case if r_σ is enabled at $f(\boldsymbol{p})$, then $f(\boldsymbol{p}) \stackrel{\sigma}{\rightarrow}$, and we are done. Otherwise, since all silent rules for J are not applicable, it must be the case that some rules in R_τ are applicable to $f(\boldsymbol{p})$, thus disabling r_σ at $f(\boldsymbol{p})$. As there are no rules above the rules in R_τ

which are applicable, the appropriate rules in R_τ are enabled at $f(\boldsymbol{p})$. Hence, $f(\boldsymbol{p}) \stackrel{\tau}{\rightarrow}$. In all cases $f(\boldsymbol{p})$ is patient. □

Example 41. We show that the language *TL* from Example 38 satisfies (14) and (15), and thus is patient. *TL* has three operators that satisfy the conditions in a non-trivial fashion. Prefixing with τ and parallel composition have non-silent rules with actions τ which are above the appropriate maximal σ-rules as described in Example 38, and no other rules for the two operators are above the mentioned non-silent rules with actions τ. Moreover, *TL* has the delay operator $\lfloor\ \rfloor(\ )$ defined in Example 31. Rule $\sigma_\emptyset$ is its maximal timed rule and the only rule above $\sigma_\emptyset$ is its silent rule τ^1. Moreover, no rule for $\lfloor\ \rfloor(\ )$ is above τ^1, Hence, (14) and (15) are satisfied.

4.4. Constancy of offers and time persistence

The constancy of offers property means informally that the passage of time does not disable and enable any visible actions: a process can perform exactly the same visible actions after any amount of time passage as it was able to perform before any time passed. Time persistence is implied by constancy of offers. It says that the passage of time does not disable visible actions: if a process can pass time and perform action a, then after any amount of time passage it can still perform a.

Theorem 42. *Let L be any timed process language and T be the LTS generated by L. If every time altering operator has only σ-rules as its timed rules or it has no timed rules at all, then T satisfies the constancy of offers property, and thus the time persistence property.*

Proof. We show that if $p \stackrel{\sigma}{\rightarrow} p'$, then we have $p \stackrel{a}{\rightarrow}$ if and only if $p' \stackrel{a}{\rightarrow}$, for all p, p' and a, by induction on the structure of process terms.

If p is a constant, say f, then all rules for f have no premises and are applicable. If f has no timed rules, then constancy of offers holds. Assume $f \stackrel{\sigma}{\rightarrow} f''$ for some f''. Suppose $f \stackrel{a}{\rightarrow} f'$ for some f' and a. In fact, $f \stackrel{a}{\rightarrow} f'$ and $f \stackrel{\sigma}{\rightarrow} f''$ are the OSOS rules with which the respective transitions $f \stackrel{a}{\rightarrow} f'$ and $f \stackrel{\sigma}{\rightarrow} f''$ are derived. If f is time preserving or it is time altering and has only σ-rules as its timed rules, then $f'' = f$. Clearly, constancy of offers holds.

Let $p = f(\boldsymbol{p})$. Assume that f is time preserving. Let $f(\boldsymbol{p}) \stackrel{\sigma}{\rightarrow} f(\boldsymbol{p}')$ be derived by a σ-rule σ_I, where $I \subseteq J$ and $J \in \mathit{Levels}(f)$. Hence, $p_i \stackrel{\sigma}{\rightarrow} p_i'$, for all $i \in I$, and $p_i' = p_i$ otherwise. Suppose $f(\boldsymbol{p}) \stackrel{a}{\rightarrow}$ is derivable by a rule r. This means that $p_j \stackrel{a_{jk}}{\rightarrow} p_{jk}$, for all $j \in \mathit{active}(r)$ and appropriate a_{jk}, and no rule in $\mathit{higher}(r)$ is applicable to $f(\boldsymbol{p})$. We easily obtain that $p_j' \stackrel{a_{jk}}{\rightarrow}$ holds for all $j \in \mathit{active}(r)$: $p_j' \stackrel{a_{jk}}{\rightarrow}$ holds by the inductive hypothesis for $j \in I \cap \mathit{active}(r)$, and $(p_j' =) p_j \stackrel{a_{jk}}{\rightarrow}$ for $j \in \mathit{active}(r) \setminus I$. As a result r is applicable to $f(\boldsymbol{p}')$. Also, no rule in $\mathit{higher}(r)$ is applicable to $f(\boldsymbol{p}')$: we easily show using the "if $p' \stackrel{a}{\rightarrow}$, then $p \stackrel{a}{\rightarrow}$" part of the inductive hypothesis that if $r' \in \mathit{higher}(r)$ is applicable to $f(\boldsymbol{p}')$, then r' is also applicable to $f(\boldsymbol{p})$ contradicting the assumption that r is enabled at $f(\boldsymbol{p})$. Hence, r is enabled at $f(\boldsymbol{p}')$, so $f(\boldsymbol{p}') \stackrel{a}{\rightarrow}$. We show that if $f(\boldsymbol{p}') \stackrel{a}{\rightarrow}$, then $f(\boldsymbol{p}) \stackrel{a}{\rightarrow}$ in a corresponding way.

If f is a time altering and has only σ-rules for its timed rules, then the proof goes correspondingly as above. Finally, if f has no timed rules at all, then $f(\boldsymbol{p}) \stackrel{\sigma}{\not\to}$, and we are done. □

General time altering operators do not preserve the two properties. Consider time altering constants f and f' with the defining rules $f \stackrel{\sigma}{\to} f'$ and $f' \stackrel{a}{\to} f$, respectively. f does not satisfy the requirements of Theorem 42 because its timed rule is not a σ-rule as f' appears in the target instead of f. Since $f \stackrel{a}{\not\to}$ the consistence of offers does not hold, but time persistence holds. Consider an additional operator g below.

$$\frac{}{g(X) \stackrel{c}{\to} g(X)}\, r_c, \quad \frac{X \stackrel{a}{\to} X'}{g(X) \stackrel{a}{\to} g(X')}\, r_a, \quad \frac{X \stackrel{\tau}{\to} X'}{g(X) \stackrel{\tau}{\to} g(X')}\, \tau_1,$$
$$\frac{X \stackrel{\sigma}{\to} X'}{g(X) \stackrel{\sigma}{\to} g(X')}\sigma_1.$$

The ordering on rules is $r_c < r_a, \tau_1$. Operator g has only one priority level $\{1\}$, and σ_1 is its only σ-rule. Clearly, g is time preserving. We derive $g(f) \stackrel{\sigma}{\to} g(f')$ and $g(f) \stackrel{c}{\to}$. The last is by r_c since the higher rule r_a is disabled at $g(f)$ as $f \stackrel{a}{\not\to}$. However, $g(f') \stackrel{a}{\to}$ and $g(f') \stackrel{c}{\not\to}$ since r_c is disabled by the higher rule r_a as $f' \stackrel{a}{\to}$. Hence, time persistence fails.

Several timed process languages satisfy the constancy of offers property, most notably a timed version of CSP described in Section 5.2.

4.5. Timelock freeness

The following condition guarantees the timelock freeness property in timed process languages.

For r_σ, the maximal timed rule for f and J, and for all rules r for f

$$\textbf{if } \mathit{active}(r) \subseteq J \textbf{ then } r_\sigma \not< r. \tag{16}$$

It requires that the maximal timed rule for the top priority level J for any operator f is not below any rule for f with active arguments in J. It is intuitive that no timed operators which satisfy (16) can block the passage of time.

Theorem 43. *Let L be any timed process language and T be the LTS generated by L. If all time altering operators have non-empty sets of timed rules, and all operators f of L, with $J = top(Levels(f))$, satisfy* (16)*, then T satisfies the timelock freeness property.*

Proof. We show timelock freeness by induction on the structure of process terms. Assume that operators have non-empty sets of defining rules. If f is a constant, then its timed rules are always applicable. In fact, the maximal time rule is enabled by (16). Hence, $f \stackrel{\sigma}{\to}$.

Consider $f(\boldsymbol{p})$. By the inductive hypothesis all its active arguments p_i can pass time, so r_σ is applicable. Moreover, condition (16) rules out the existence of any rule for f with active arguments in J which is above r_σ. Hence, r_σ is enabled at $f(\boldsymbol{p})$, and $f(\boldsymbol{p}) \stackrel{\sigma}{\to}$. □

The timelock freeness property is not so useful. It does not hold in most timed process languages that satisfy maximal progress, urgency or patience. From this point of view, a

less restrictive version, the discussed above weak timelock freeness, is more suitable as it can coexist with both the maximal progress and patience properties.

5. Representing timed process operators in our framework

In this section we examine several existing process languages with time and investigate how they fit into our framework. Some languages, for example TPL [27], can be 'directly' represented as our timed process languages. Others, such as ACP^{drt} [9,10], do not have adequate reformulation due to, for example, the lack of predicates in the OSOS approach. The main goal of this section is to

- investigate the operational definitions of the standard and new timed operators in the considered process languages with time, and to see how they fit into our framework,
- examine whether or not the timed properties hold in the considered process languages by checking the validity of the proposed conditions from Section 4,
- investigate the timed extensions of several process languages which have *strong* semantics (e.g. strong bisimulation) by applying our approach. This produces timed process languages with *weak* semantics.

The process languages with time that we consider can be put into several groups depending how closely they fit into our framework:

- TPL and the discrete relative time version of CSP: the operational semantics can be equivalently represented in our framework as informally defined in Section 2.3.
- ATP: the SOS rules for the operators can be expressed in the OSOS format. However, since ATP uses strong bisimulation as semantics the reformulation of ATP as a timed process language requires the addition of silent rules, which are not in the original ATP.
- ET-LOTOS and the discrete relative time versions of CCS: these process languages use a family of relations $\xrightarrow{t}$, for $t \in \mathbb{N}$, to represent the passage of t time units. We only consider their versions for $t = 1$.
- ACP^{drt} with discrete relative timing does not fit so well within our framework. We do not allow predicates and negated predicates: these are used widely in ACP^{drt}. Also, the specific form of silent rules we use means that auxiliary operators such left merge are not representable.

5.1. TPL

TPL [27] is a timed extension of CCS with two delay operators. Simple delay σX is defined by $\sigma X \xrightarrow{\sigma} X$ and is clearly time altering. The mentioned earlier $\lfloor\ \rfloor(\ )$ (it shares the notation with an operator of ACP given below) is defined as follows:

$$\frac{X \xrightarrow{\alpha} X'}{\lfloor X \rfloor(Y) \xrightarrow{\alpha} X'}, \qquad \frac{X \xrightarrow{\tau}}{\lfloor X \rfloor(Y) \xrightarrow{\sigma} Y}.$$

In Example 31 we have shown how $\lfloor\ \rfloor(\ )$ can be redefined as a time altering operator. Prefixing with visible action is defined by $a.X \xrightarrow{a} X$ and the σ-rule $a.X \xrightarrow{\sigma} a.X$. Prefixing with τ is defined by the single rule $\tau.X \xrightarrow{\tau} X$; hence, it is urgent and time altering since it has no σ-rule. The SOS rules for the deadlocked operator $\mathbf{0}$ and the choice operator make them time preserving. The LTS for TPL is time deterministic and maximal progress

holds because of the prefixing with τ and the following timed rule for the parallel operator (in addition to the standard CCS rules).

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

We have seen in Examples 38 and 41 that TPL can be equivalently expressed as the timed process language *TL*. By inspecting the SOS rules for TPL and *TL* we notice that the resulting LTSs are the same. We easily check that LTS for *TL* satisfies the patience and weak timelock freeness properties in addition to time determinacy and maximal progress. However, time persistence and consistency of offers do not hold: $\lfloor a \rfloor(b) \xrightarrow{a}$ but $\lfloor a \rfloor(b) \xrightarrow{\sigma} b \xrightarrow{a}$.

Our congruence results imply that timed rooted eager bisimulation preorder is a precongruence for *TL* and, thus, for TPL. This is in addition to *barbs* semantics [27] which apply to TPL and to *TL*.

5.2. CSP with time

The most popular extensions of CSP with time use the dense time domain [42,44,45]. However, we can construct a discrete time version of CSP by altering the operational definitions of the dense time CSP. A comprehensive exposition of (dense) Timed CSP is given by Schneider in [45]; it contains operational semantics and a discussion concerning timed properties of Timed CSP.

There are many ways discrete time can be added to CSP. Our approach is close in spirit to that taken by Roscoe in [43]. The passage of time can be introduced in the language by several means. The deadlocked process "*STOP*" has the single timed rule $STOP \xrightarrow{\sigma} STOP$ and, hence, is time preserving. The successful termination process "*SKIP*" has two SOS rules: $SKIP \xrightarrow{\surd} STOP$ and $SKIP \xrightarrow{\sigma} SKIP$; hence, it is also time preserving. Prefixing is time preserving and it has the usual σ-rule. We have a family of delay processes $WAIT\,d$ with $d \geqslant 0$. "$WAIT\,d$" is defined as the process $STOP \overset{d}{\triangleright} SKIP$, where "$\overset{d}{\triangleright}$" is a member of a family of timeout operators defined as follows:

$$\frac{X \xrightarrow{a} X'}{X \overset{d}{\triangleright} Y \xrightarrow{a} X'}, \quad \frac{X \xrightarrow{\tau} X'}{X \overset{d}{\triangleright} Y \xrightarrow{\tau} X' \overset{d}{\triangleright} Y}, \quad \frac{X \xrightarrow{\sigma} X'}{X \overset{d+1}{\triangleright} Y \xrightarrow{\sigma} X' \overset{d}{\triangleright} Y}, \quad \frac{}{X \overset{0}{\triangleright} Y \xrightarrow{\tau} Y}.$$

In our terminology the timeout operators are time altering since the timed rules for $\overset{d+1}{\triangleright}$ are not σ-rules, and since $\overset{0}{\triangleright}$ has no timed rules at all. The operators are τ-preserving. The external choice operator has the usual σ-rule that requires that both arguments synchronise the passage of time. The internal choice operator has no timed rules thus making it time altering. Several types of parallel composition operators have the usual σ-rules.

One of the timed properties that Timed CSP satisfies is maximal progress. We show how our reformulation of the discrete relative version of Timed CSP satisfies maximal progress. As there is no prefixing with silent actions this property is guaranteed by careful definitions of three operators that introduce silent actions. They are the internal choice, the hiding and the sequential composition operators. The internal choice is defined by the two rules $X \sqcap Y \xrightarrow{\tau} X$ and $X \sqcap Y \xrightarrow{\tau} Y$. The lack of timed rules for $\sqcap$ ensures that it is time altering and that it preserves maximal progress. The hiding is defined by the usual rules together with the timed rule

$$\frac{X \xrightarrow{\sigma} X' \quad \{X \overset{a}{\not\to} \mid a \in A\}}{X \setminus A \xrightarrow{\sigma} X' \setminus A}.$$

In our framework this rule can be represented as a σ-rule

$$\frac{X \xrightarrow{\sigma} X'}{X \setminus A \xrightarrow{\sigma} X' \setminus A}$$

that is placed below the two standard rules with the action τ for the hiding operator. Note that we do not place the above σ-rule below the action rules for $\setminus$ with actions $b \notin A$. Overall, the hiding operator is time altering. The sequential composition can be defined in our setting by the following rules, where $\alpha \neq \surd$,

$$\frac{X \xrightarrow{\alpha} X'}{X; Y \xrightarrow{\alpha} X'; Y}\, r_\alpha, \qquad \frac{X \xrightarrow{\surd} X'}{X; Y \xrightarrow{\tau} Y}\, r_\surd, \qquad \frac{X \xrightarrow{\sigma} X'}{X; Y \xrightarrow{\sigma} X'; Y}\, \sigma_1$$

together with the ordering $\sigma_1 < r_\surd, r_\tau$. The above three operators are τ-preserving and time altering. Our definitions of these operators satisfy conditions (10) and (11), so maximal progress holds by Theorem 37. Also, we easily check that (14) and the conditions in Theorem 40 are satisfied, so patience and weak timelock freeness both hold. Finally, the constancy of offers and the persistence properties are satisfied by Theorem 42.

Our congruence results imply that timed rooted eager bisimulation preorder is a precongruence for the presented version of TCSP. This is in addition to a version of (discrete relative time) semantics that TCSP has.

5.3. Algebra of Timed Processes

The passage of time in *Algebra of Timed Processes* (ATP) of Nicolin and Sifakis [39] is introduced by the *unit-delay* operator $\lfloor\ \rfloor(\ )$ (not to be confused with the above delay operator of TPL). The operator is defined in our framework by the following SOS rules:

$$\frac{X \xrightarrow{\alpha} X'}{\lfloor X \rfloor(Y) \xrightarrow{\alpha} X'}, \qquad \lfloor X \rfloor(Y) \xrightarrow{\sigma} Y.$$

We easily see that unit-delay is time altering. ATP has urgent prefixing and urgent **0** which are both time altering. However, the choice, parallel and encapsulation operators are time preserving. Furthermore, [39] introduces four families of other timed operators: the *timeout* at d, the *start delay* within d, the *unbounded start delay* and the *execution delay* within d, where $d \in \mathbb{N}$. These operators can be expressed in the OSOS framework by families of operators as follows:

The family of timeout operators "$\cdot \rhd^d \cdot$", where $d > 0$, is defined as follows:

$$\frac{X \xrightarrow{\alpha} X'}{X \rhd^d Y \xrightarrow{\alpha} X'}, \qquad \frac{X \xrightarrow{\sigma} X'}{X \rhd^{d+1} Y \xrightarrow{\sigma} X' \rhd^d Y}, \qquad \frac{X \xrightarrow{\sigma} X'}{X \rhd^1 Y \xrightarrow{\sigma} Y}.$$

The family of start delay operators "$\lfloor\cdot\rfloor^d\cdot$", where $d > 0$, has the the following defining rules, and the ordering is $\sigma_\emptyset^{d+1} < \sigma_1^{d+1}$ and, by condition (6) as $\lfloor\cdot\rfloor^{d+1}\cdot$ is τ-sensitive, $\sigma_\emptyset^{d+1} < r_\tau$.

$$\frac{X \xrightarrow{\alpha} X'}{\lfloor X \rfloor^d Y \xrightarrow{\alpha} X'}\, r_\alpha, \qquad \frac{}{\lfloor X \rfloor^1 Y \xrightarrow{\sigma} Y}\, \sigma,$$

$$\frac{X \xrightarrow{\sigma} X'}{\lfloor X \rfloor^{d+1} Y \xrightarrow{\sigma} \lfloor X' \rfloor^{d} Y}\ \sigma_1^{d+1}, \qquad \frac{}{\lfloor X \rfloor^{d+1} Y \xrightarrow{\sigma} \lfloor X \rfloor^{d} Y}\ \sigma_{\emptyset}^{d+1}.$$

The family of unbounded start delay operators "$\lfloor \cdot \rfloor^{\omega}$" can be redefined in our setting by the following rules with the ordering $\sigma_\emptyset < \sigma_1$ and $\sigma_\emptyset < r_\tau$.

$$\frac{X \xrightarrow{\alpha} X'}{\lfloor X \rfloor^{\omega} \xrightarrow{\alpha} X'}\ r_\alpha, \quad \frac{X \xrightarrow{\sigma} X'}{\lfloor X \rfloor^{\omega} \xrightarrow{\sigma} \lfloor X' \rfloor^{\omega}}\ \sigma_1, \quad \frac{}{\lfloor X \rfloor^{\omega} \xrightarrow{\sigma} \lfloor X \rfloor^{\omega}}\ \sigma_\emptyset.$$

Finally, the family of execution delay operators "$\lceil \cdot \rceil^{d} \cdot$", $d > 0$, is defined by unordered rules:

$$\frac{X \xrightarrow{\alpha} X'}{\lceil X \rceil^{d} Y \xrightarrow{\alpha} \lceil X' \rceil^{d} Y}\ r_*, \quad \frac{X \xrightarrow{\sigma} X'}{\lceil X \rceil^{1} Y \xrightarrow{\sigma} Y}\ \sigma, \quad \frac{X \xrightarrow{\sigma} X'}{\lceil X \rceil^{d+1} Y \xrightarrow{\sigma} \lceil X' \rceil^{d} Y}\ \sigma_1.$$

Note that the first three families of operators are not time altering because τ-sensitive operators appear in the targets of some rules. The last family consists of time altering operators.

The LTS generated by ATP is time deterministic but not timelock free due to urgent prefixing and **0**. Clearly, the urgency property holds. The maximal progress property does not hold since parallel composition of two processes may pass time even when communication is possible. Also, time persistence does not hold: $\lfloor a \rfloor(b) \xrightarrow{a}$ but $\lfloor a \rfloor(b) \xrightarrow{\sigma} b \xrightarrow{a\!\!\!/}$.

5.4. CCS with time

One of the first extensions of CCS with time was developed by Wang [58,59]. He used real time domain and defined operational semantics for the extended language. Based on this work, we can easily define a simple discrete relative time version Wang's process language.

In the mentioned extension of CCS prefixing with visible actions is time preserving but prefixing with τ is urgent: there is no timed rule for prefixing with τ. In our setting we define this by putting $\tau.X \xrightarrow{\sigma} \tau.X$ below the usual rule $\tau.X \xrightarrow{\tau} X$. Thus, prefixing with τ is time altering. The passage of time is introduced by a family of delay operators $\epsilon(d)$, for $d \geqslant 0$, with the following defining rules:

$$\frac{X \xrightarrow{\sigma} X'}{\epsilon(d+1).X \xrightarrow{\sigma} \epsilon(d+1).X'}, \quad \frac{}{\epsilon(d+1).X \xrightarrow{\sigma} \epsilon(d).X}, \quad \frac{X \xrightarrow{\chi} X'}{\epsilon(0).X \xrightarrow{\chi} X'}.$$

Examining the rules above, we notice the delay operators have one priority level {1}. Since the second and third rules above are not σ-rules, the operators are time altering.

The deadlocked operator has one σ-rule, and the choice operator has also only one σ-rule, namely $\sigma^{+}_{\{1,2\}}$ from Example 28. The two operators are time preserving. The parallel composition operator $\|$ allows time to pass provided that the component processes can pass time and that communication is not possible. The last requirement is represented by a side condition that checks the initial actions of the components for possible communication. In our formalism this is done purely with OSOS rules: we place the communication rule and the two τ-rules above the sigma rule $\sigma^{\|}_{\{1,2\}}$ as done in Section 4.1. This makes the parallel a time altering operator. As a result, condition (11) is satisfied and maximal progress is preserved in the resulting LTS. Also, conditions (14) and (15), and those in Theorem 40, are satisfied by the prefixing with τ and parallel operators. Hence, patience and weak timelock freeness hold. Clearly, time determinacy also is satisfied. Although the delay operators

do not satisfy the conditions from Theorem 42 this timed version of CCS satisfies time persistence but not constancy of offers: $\epsilon(1).a.\mathbf{0} \xrightarrow{\sigma} \epsilon(0).a.\mathbf{0} \xrightarrow{a}$ and $\epsilon(1).a.\mathbf{0} \overset{a}{\not\to}$.

Timed rooted eager bisimulation is a precongruence in the above described discrete relative time version of Wang's timed CCS.

Moller and Tofts [34] defined another timed extension of CCS called *Temporal* CCS (TCCS). Unlike in ATP and TPL that we have described above, they used a family of delay operators (t): $(t).P$ behaves like P after t time units. These operators are essentially time altering operators. Similarly as in ATP, there is urgent prefixing and $\mathbf{0}$ does not let time pass. TCCS has two choice operators instead of a general delay operator. It has a *strong* choice, which is essentially the time preserving + of CCS, and a *weak* choice $\oplus$. The latter resolves the choice either internally or by the argument with a longer initial delay. A similar operator can be defined in our framework by the usual action rules for $\oplus$ and the σ-rules

$$\frac{X \xrightarrow{\sigma} X'}{X \oplus Y \xrightarrow{\sigma} X'}\,\sigma_X, \qquad \frac{Y \xrightarrow{\sigma} Y'}{X \oplus Y \xrightarrow{\sigma} Y'}\,\sigma_Y, \qquad \frac{X \xrightarrow{\sigma} X' \quad Y \xrightarrow{\sigma} Y'}{X \oplus Y \xrightarrow{\sigma} X' \oplus Y'}\,\sigma_{XY},$$

with $\sigma_X, \sigma_Y < \sigma_{XY}$. The LTS for TCCS is time deterministic, but due to urgent prefixing and $\oplus$ it fails timelock freeness, maximal progress and constancy of offers: $(\sigma).a.\mathbf{0} \oplus \mathbf{0} \xrightarrow{\sigma} a.\mathbf{0} \xrightarrow{a}$ but $(\sigma).a.\mathbf{0} \oplus \mathbf{0} \overset{a}{\not\to}$.

In [35], a version of TCCS called *l*TCCS is proposed. Apart from the mentioned delay operators, *l*TCCS has only time preserving versions of CCS operators. Hence, its LTS satisfies time determinacy, patience, weak timelock freeness and timelock freeness by Theorems 43. Also, time persistence holds.

5.5. ET-LOTOS

ET-LOTOS [30,31] is an extension of LOTOS with data types and an abstract time domain. We consider a version of ET-LOTOS with discrete relative time, and see how to reformulate it in our framework.

Since we use $\xrightarrow{\sigma}$ transitions and not $\xrightarrow{d}$ transitions as in ET-LOTOS, where $d \in \mathbb{N}$ represents a number of time units, we redefine ET-LOTOS operators using $\xrightarrow{\sigma}$. The operator "**stop**" is a time preserving operator with the following rule.

$$\frac{}{\mathbf{stop} \xrightarrow{\sigma} \mathbf{stop}}.$$

Operator "**block**" is a time altering operator as it has no defining rules. Delay prefixing "Δ^d" operators, $d \in \mathbb{N}$, can be defined in our framework by a family of operators as follows:

$$\frac{}{\Delta^d X \xrightarrow{\sigma} \Delta^{d-1} X} \quad d > 1, \qquad \frac{}{\Delta^1 X \xrightarrow{\sigma} X}.$$

Prefixing with visible actions operators are essentially timed rebo although we cannot model directly the *satisfaction predicates* of ET-LOTOS. The prefixing with silent action, "$\mathbf{i}\{d\}; \cdot$", where $\{d\}$ is a *life reducer*, models a form of urgency of silent actions: τ will occur within the next d time units, and $\mathbf{i}\{d\}; p$ will not idle for more than d time units. We define $\mathbf{i}\{d\}; \cdot$ as follows:

$$\frac{}{\mathbf{i}\{d\}; X \xrightarrow{\tau} X}, \qquad \frac{}{\mathbf{i}\{d\}; X \xrightarrow{\sigma} \mathbf{i}\{d-1\}; X} \quad d > 0.$$

The parallel composition, choice and disabling operators of ET-LOTOS are essentially timed rebo: in fact they are time preserving operators.

The operator "**hide**" is important for the *maximal progress of hidden actions* property, which states that any hidden action is urgent, i.e. it must happen without a delay. Our definition of **hide** employs an ordering on the defining rules instead of negative premises and *lookahead* [31]. The two usual rules are give below, where α is any action not in A and a is any visible action in A.

$$\frac{X \xrightarrow{\alpha} X'}{\mathbf{hide} A \mathbf{\ in\ } X \xrightarrow{\alpha} \mathbf{hide}\ A \mathbf{\ in\ } X'}, \qquad \frac{X \xrightarrow{a} X'}{\mathbf{hide}\ A \mathbf{\ in\ } X \xrightarrow{\tau} \mathbf{hide}\ A \mathbf{\ in\ } X'} \quad r_a.$$

The timed rule is simply the σ-rule for the only argument of **hide**:

$$\frac{X \xrightarrow{\sigma} X'}{\mathbf{hide}\ A \mathbf{\ in\ } X \xrightarrow{\sigma} \mathbf{hide}\ A \mathbf{\ in\ } X'} \quad \sigma_1.$$

The ordering is $\sigma_1 < r_a$ for all $a \in A$. We check that **hide** $\{a\}$ **in** $\Delta^3 a$; **stop** can idle for at most 3 units of time before it offers τ, although $\Delta^3 a$; **stop** can idle forever.

Strictly speaking, the above formulation of the hide operator is not time altering. This is because the ordering on its rules does not satisfy condition (7) as required in part 2 of Definition 30. Since $\sigma_1 < r_a$, for all $a \in A$, the definition requires $\sigma_1 < \tau_1$ as well. The lack of this ordering makes the operator unsafe with respect to our timed preorder $\sqsubseteq_{tr}$. To illustrate this consider a timed process language with time preserving versions of the CCS **0** and action prefixing as well as with the above hide operator. We have $p = \tau.\tau.a.\mathbf{0} \sqsubseteq_{tr} \tau.a.\mathbf{0} = q$ but **hide** $\{a\}$ **in** $p \not\sqsubseteq_{tr}$ **hide**$\{a\}$ **in** q. The reasons for this is as follows: On one hand, **hide**$\{a\}$ **in** $p \xrightarrow{\tau}$ **hide**$\{a\}$ **in** $\tau.a.\mathbf{0}$ by the τ-rule, i.e. the first rule above with $\alpha = \tau$. Then, **hide**$\{a\}$ **in** $\tau.a.\mathbf{0} \xrightarrow{\sigma}$ by the rule σ since it is not disabled by the τ-rule. On the other hand, **hide**$\{a\}$ **in** $q \xrightarrow{\tau}$ **hide**$\{a\}$ **in** $a.\mathbf{0}$, and since r_a is enabled at **hide**$\{a\}$ **in** $a.\mathbf{0}$ the rule σ is disabled, so **hide**$\{a\}$ **in** $a.\mathbf{0} \xrightarrow{\sigma}$.

Overall, our reformulation of ET-LOTOS is time deterministic and urgent. However, as seen above it does not preserve the chosen semantics based on the timed rooted eager bisimulation. Moreover, both our reformulation of ET-LOTOS and ET-LOTOS itself does not preserve the standard weak bisimulation [31]; an appropriate weak equivalence based on bisimulation, which is also a congruence for ET-LOTOS, is still to be found. It would be interesting to investigate if a reformulation of (a large part of) ET-LOTOS in terms of timed rebo operators would alleviate the problem, and deliver a weak timed congruence for ET-LOTOS.

5.6. ACP with discrete relative time

Out of many extensions of ACP [11] with discrete time that have been proposed over the years we shall consider ACP^{drt}, ACP *with relative discrete timing*, one of the three different flavours of ACP with discrete time developed by Baeten, Bergstra and Middelburg in [7,9,10].

The operational rules for some ACP operators do not fit directly into the OSOS format, they fit into a more general format, called *panth* [56], that allows predicates and negative premises. However, with some encoding of ACP operators one can give ACP a satisfactory OSOS semantics. [1] Alternatively, one could extend the OSOS approach to include

[1] One would have to encode the predicates for successful termination and immediate catastrophic termination by the means of special constants and actions. This would have to be followed by a change of several axioms and the definition of timed (strong) bisimulation to reflect the alterations made to the syntax.

predicates, and perhaps other features such as lookahead, then find a subformat of the extended format for eager bisimulation or branching bisimulation, and finally investigate how ACP^{drt} fits in. This we shall leave to future research.

In the remainder of this subsection we examine some of the operators of ACP^{drt} to see if they can be expressed in our formalism. ACP^{drt} is an extension of ACP with several timed and untimed operators. Since in ACP$_{drt}$ there no distinction between visible and silent actions (timed strong bisimulation is used as semantics) we pay no attention to silent rules. ACP$_{drt}$ has urgent actions $cts(a)$ (or $\underline{a}$ in [5,9]) that can be defined in our setting by the single rule $cts(a) \xrightarrow{a} \surd$, where $\surd$ is a constant denoting a successful termination. Thus, operators $cts(a)$ are time altering. The language has non-urgent actions defined by the two rules $a \xrightarrow{a} \surd$ and $a \xrightarrow{\sigma} a$, hence making action prefixing time preserving operators. Note that the last form of action prefixing is not present in [9]. There are two deadlock operators, δ and $\dot{\delta}$, first being time preserving operator with the timed rule $\delta \xrightarrow{\sigma} \delta$, and the second being time altering with no timed rules and representing immediate and catastrophic deadlock. The passage of time is introduced with the *relative discrete time unit delay* operators σ^m_{rel} for $m \in \mathbb{N}$.

The choice operator $+$ has the timed rules as in Example 29 which make it time altering. The timed rules for the merge operator "$\|$" and the communication operator "$|$", which are given below, make them time preserving.

$$\frac{X \xrightarrow{\sigma} X' \quad Y \xrightarrow{\sigma} Y'}{X \parallel Y \xrightarrow{\sigma} X' \parallel Y'}, \qquad \frac{X \xrightarrow{\sigma} X' \quad Y \xrightarrow{\sigma} Y'}{X \between Y \xrightarrow{\sigma} X' \between Y'}, \qquad \frac{X \xrightarrow{\sigma} X' \quad Y \xrightarrow{\sigma} Y'}{X \mid Y \xrightarrow{\sigma} X' \mid Y'}.$$

In ACP [11], the second argument of left-merge is not active (wild). However, in the considered timed version of ACP the second argument of $\between$ is tested in the premises of rules that define immediate deadlock for $\between$. Thus, there is $Y \xrightarrow{\sigma} Y'$ in the premises of the σ-rule for the left-merge above. Overall, ACP^{drt} is time deterministic and satisfies the urgency property.

6. Conclusion and future work

We have proposed a uniform framework for defining timed process languages which are compositional with respect to timed rooted eager bisimulation and also satisfy some common properties of time passage such as time determinacy. As a result many process operators, both standard and timed, can be represented in our framework. Some process languages with discrete relative time such as TPL can be reformulated equivalently in our framework, others such as ACP^{drt} require more powerful SOS rules. When a process language can be reformulated in the proposed framework of OSOS rules, we can verify if it preserves $\sqsubseteq_{tr}$ and is time deterministic. This is done by checking whether or not the ordered rules for the language's operators satisfy several syntactical constraints.

We have also given several other conditions such that maximal progress, urgency, patience, weak timelock freeness, timelock freeness, and time persistence and constancy of offers are preserved.

In future we hope to generalise our results to an abstract time domain, which would cover both discrete and dense time, and to extend the OSOS format with predicates. The latter result would permit direct representation of SOS rules for ACP^{drt}, and may lead to a variant with weak timed semantics. This may also lead to a new syntactical method for checking whether or not certain timed properties hold in a timed process language.

Acknowledgements

We would like to thank Iain Phillips, Matthew Hennessy and Rance Cleaveland for discussions concerning process languages with time. The first author was supported by the University of Leicester Research Fund grant, and also partially supported by the Ministry of Education, Science and Culture of Japan grant 09780270 and the COE program on Intelligent Media Integration for Social Information Infrastructure at Nagoya University. The second author was supported by the Ministry of Education, Science and Culture of Japan grant 13680408 and the JSPS Joint Research Project between Japan and the United Kingdom.

Appendix

A. Congruence theorem for rooted eager bisimulation

The proof of Theorem 22 follows the same pattern as the proofs of the congruence results for branching bisimulation and eager bisimulation and for bbo and ebo process languages, respectively, in [53], as well as the proof of the congruence result for the rooted branching bisimulation in [18]. The presented rebo process languages are extensions of ebo process languages with τ-sensitive operators like the CCS+.

Firstly, we introduce an alternative characterization of an eager bisimulation in Proposition 46. It says that eager bisimilar processes have matching behaviour represented in terms of actions as well as *action refusals*. Recall that action refusal is the inability to perform an action in a stable state [40]. More precisely, process p refuses action α if p is stable and p cannot perform α, namely $p \stackrel{\tau}{\not\rightarrow} \stackrel{\alpha}{\not\rightarrow}$. We will also use the following notion of a process variable being emph τ-preserving in a context.

Definition 44. Let $G = (\Sigma, \mathsf{Act}_\sigma, R, <)$ be a rebo process language. A process variable X_i is τ-preserving in X_i, for every variable X_i; and X_i is τ-preserving in Σ context $f(t_1, \ldots, t_n)$ if, for each $j \in \{1, \ldots, n\}$, whenever $X_i \in var(t_j)$ and $j \in active(f)$, then $\tau_j^f \in rules(f)$ and X_i is τ-preserving in t_j.

Proposition 45. *Let $G = (\Sigma, \mathsf{Act}_\sigma, R, <)$ be a* rebo *process language. If a context over G contains only τ-*preserving *operators, then every variable of the context is τ-*preserving *in the context.*

Notation. In this section we shall abuse slightly the notation by treating σ as one of the visible actions. Hence, when we write a, b_{ij}, etc. we mean that these actions are either visible or σ.

Proposition 46. *Given $(\mathcal{P}, \mathsf{Act}_\sigma, \rightarrow)$, a relation $E \subseteq \mathcal{P} \times \mathcal{P}$ is an eager bisimulation if and only if, for all p and q such that pEq, the following properties hold.*

$(E.a')$ $\forall p', \chi.\ [\ p \stackrel{\chi}{\rightarrow} p'$ *implies* $\exists q', q''.\ (q \stackrel{\tau}{\Rightarrow} q' \stackrel{\widehat{\chi}}{\rightarrow} q''$ *and* $p'Eq'')]$

$(E.b')$ $p \stackrel{\tau}{\not\rightarrow}$ *implies* $q\Downarrow$

$(E.c')$ $\forall q', \chi.\ [\ p\Downarrow$ *and* $q \stackrel{\chi}{\rightarrow} q'$ *implies* $\exists p', p''.\ (p \stackrel{\tau}{\Rightarrow} p' \stackrel{\widehat{\chi}}{\rightarrow} p''$ *and* $p''Eq')]$

$(E.a'')$ $p \stackrel{\tau}{\not\to}$ *implies* $\forall q'.[q \stackrel{\tau}{\Rightarrow} q'$ *implies* $(pEq'$ *and*
$(E.a^{\star})$ $\forall a, p'.(p \stackrel{a}{\to} p'$ *implies* $\exists q'', q'''.(q' \stackrel{\tau}{\Rightarrow} q'' \stackrel{a}{\to} q'''$ *and* $p'Eq'''))$
$(E.c^{\star})$ $\forall a, q''.(q' \stackrel{a}{\to} q''$ *implies* $\exists p'.(p \stackrel{a}{\to} p'$ *and* $p'Eq'')))]$
$(E.c'')$ $p\Downarrow$ *and* $q \stackrel{\tau}{\not\to}$ *implies* $\forall p'.[p \stackrel{\tau}{\Rightarrow} p'$ *implies* $(p'Eq$ *and*
$(E.a^{\dagger})$ $\forall a, p''.(p' \stackrel{a}{\to} p''$ *implies* $\exists q'.(q \stackrel{a}{\to} q'$ *and* $p'''Eq'))$
$(E.c^{\dagger})$ $\forall a, q'.(q \stackrel{a}{\to} q'$ *implies* $\exists p'', p'''.(p' \stackrel{\tau}{\Rightarrow} p'' \stackrel{a}{\to} p'''$ *and* $p'''Eq')))]$.

Proof. For the "if" part we only need to show that $p\Downarrow$ implies $q\Downarrow$. Suppose for a contradiction that $p\Downarrow$ and $q\Uparrow$. Since $q\Uparrow$ there exists the following infinite sequence from q: $q \stackrel{\tau}{\to} q_1 \stackrel{\tau}{\to} q_2 \stackrel{\tau}{\to} \cdots$. As $p\Downarrow$, $q \stackrel{\tau}{\to} q_1$ implies, by $(E.c')$, $p \stackrel{\tau}{\Rightarrow} p' \stackrel{\widehat{\tau}}{\to} p_1$ and p_1Eq_1 for some p' and p_1. Since p_1 is a τ-derivative of p, $p_1\Downarrow$. Again, by $(E.c')$, $q_1 \stackrel{\tau}{\to} q_2$ implies $p_1 \stackrel{\tau}{\Rightarrow} p_1' \stackrel{\widehat{\tau}}{\to} p_2$ with p_2Eq_2. By repeating this derivation process we can construct the following sequence: $p \stackrel{\tau}{\Rightarrow}\stackrel{\widehat{\tau}}{\to} p_1 \stackrel{\tau}{\Rightarrow}\stackrel{\widehat{\tau}}{\to} p_2 \stackrel{\tau}{\Rightarrow}\stackrel{\widehat{\tau}}{\to} \cdots$, where p_nEq_n for all n. Since $p\Downarrow$, there exists p_k such that $p_l = p_k$, for all $k \leqslant l$, and $p_k \stackrel{\tau}{\not\to}$. But we still have $q_k\Uparrow$. This contradicts $(E.b')$.

Now we consider the "only if" part. Since $p \stackrel{\tau}{\not\to}$ implies $p\Downarrow$, the property $(E.b')$ holds by $(E.b)$. Next, consider $(E.a'')$. By repeatedly applying $(E.c)$ to q and its τ-derivatives we easily see that $\forall q'.\ (q \stackrel{\tau}{\Rightarrow} q'$ implies $pEq')$ holds. Properties $(E.a^{\star})$ and $(E.c^{\star})$ follow easily by $(E.a)$ and $(E.c)$. The property $(E.c'')$ is shown similarly. □

Now, we are ready to prove Theorem 22. Let $G = (\Sigma, \mathsf{Act}, R, <)$ be a rebo process language and let $\to$ be the transition relation generated by G as in Definition 13. We start by defining relations $\mathcal{R}$ and $\mathcal{E}$ over $T(\Sigma) \times T(\Sigma)$. $\mathcal{R}$ is the least relation that satisfies

- $\boldsymbol{u}\mathcal{R}\boldsymbol{v}$ if $\boldsymbol{u} \stackrel{\sqsubset}{\sim}_r \boldsymbol{v}$, and
- $C[\boldsymbol{u}]\mathcal{R}C[\boldsymbol{v}]$ if $\boldsymbol{u}\mathcal{R}\boldsymbol{v}$,

where $C[\boldsymbol{X}]$ is any Σ context and $\boldsymbol{u}$ and $\boldsymbol{v}$ are vectors of the right length of closed Σ terms. With the same notation, relation $\mathcal{E}$ is the least relation that satisfies

- $\boldsymbol{u}\mathcal{E}\boldsymbol{v}$ if $\boldsymbol{u} \stackrel{\sqsubset}{\simeq} \boldsymbol{v}$, and
- $C[\boldsymbol{u}]\mathcal{E}C[\boldsymbol{v}]$ if $u_i\mathcal{E}v_i$, whenever X_i is τ-preserving in $C[\boldsymbol{X}]$, and $u_i\mathcal{R}v_i$ otherwise, for each i.

It is easy to show that $\mathcal{R}$ is the least congruence relation which contains $\stackrel{\sqsubset}{\sim}_r$. As for $\mathcal{E}$, we need to be sensitive to non-τ-preserving occurrences of variables in contexts when defining $\mathcal{E}$. This is to avoid having $\tau.a.\mathbf{0} + b.\mathbf{0}\mathcal{E}a.\mathbf{0} + b.\mathbf{0}$ due to $\tau.a.\mathbf{0}\mathcal{E}a.\mathbf{0}$. We have the following result: for every $f \in \Sigma$ and appropriate vectors of process terms $\boldsymbol{u}$ and $\boldsymbol{v}$, if $u_i\mathcal{E}v_i$, for all i such that $\tau_i \in \mathit{rules}(f)$, and $u_i\mathcal{R}v_i$ otherwise, then $f(\boldsymbol{u})\mathcal{E}f(\boldsymbol{v})$. So, since both arguments of $+$ are not τ-preserving we require $u_i\mathcal{R}v_i$, for $i \in \{1, 2\}$, in order to have $u_1 + u_2\ \mathcal{E}\ v_1 + v_2$.

All we need to do now is to show that $\mathcal{R}$ is a rooted eager bisimulation relation and that $\mathcal{E}$ is an eager bisimulation relation. We prove in parallel the following two statements by induction of the depth of process terms.

Statement 1. If $p\mathcal{R}q$, then p, q satisfy $(R.a)$ and $(R.c)$.

Statement 2. If $p\mathcal{E}q$, then p, q satisfy $(E.a')$, $(E.b')$, $(E.c')$, $(E.a'')$ and $(E.c'')$.

Note that the properties mentioned in Statement 1 were defined in Definition 4 and the properties from Statement 2 were defined in Proposition 46.

Statement 1. Assume $p\mathcal{R}q$ and that properties $(R.a)$ and $(R.c)$ hold for all subterms of p and q that are related by $\mathcal{R}$. If $p \sqsubseteq_r q$, then we are done. Else, p and q can be represented as $f(\boldsymbol{u})$ and $f(\boldsymbol{v})$ respectively for some f, $\boldsymbol{u}$ and $\boldsymbol{v}$ with $\boldsymbol{u}\mathcal{R}\boldsymbol{v}$. Instead of showing that properties $(R.a)$ and $(R.c)$ hold for $f(\boldsymbol{u})$ and $f(\boldsymbol{v})$, we prove the following stronger properties $(\mathcal{R}.a)$ and $(\mathcal{R}.c)$.

$$f(\boldsymbol{u}) \xrightarrow{\chi} u' \quad \text{implies } \exists C[V], D[\boldsymbol{X}, \boldsymbol{Y}], \boldsymbol{u}^\dagger, \boldsymbol{u}', \boldsymbol{v}^\dagger, \boldsymbol{v}', \boldsymbol{v}'', v''.(f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} C[\boldsymbol{v}'] \xrightarrow{\chi} v'' \text{ and } v'' = D[\boldsymbol{v}^\dagger, \boldsymbol{v}''] \text{ and } u' = D[\boldsymbol{u}^\dagger, \boldsymbol{u}'] \text{ and } u' \mathcal{E} v'')$$

$$f(\boldsymbol{v}) \xrightarrow{\chi} v' \quad \text{and } f(\boldsymbol{u})\Downarrow \text{ implies } \exists C[V], D[\boldsymbol{X}, \boldsymbol{Y}], \boldsymbol{v}^\dagger, \boldsymbol{v}', \boldsymbol{u}^\dagger, \boldsymbol{u}', \boldsymbol{u}'', u''. (f(\boldsymbol{u}) \stackrel{\tau}{\Rightarrow} C[\boldsymbol{u}'] \xrightarrow{\chi} u'' \text{ and } u'' = D[\boldsymbol{u}^\dagger, \boldsymbol{u}''] \text{ and } v' = D[\boldsymbol{v}^\dagger, \boldsymbol{v}'] \text{ and } u'' \mathcal{E} v')$$

Property $(\mathcal{R}.\mathbf{a})$: Assume $f(\boldsymbol{u}) \xrightarrow{\chi} u'$. This means that there is an action rule (the actions in the premises may be σ) or a τ-rule or a silent choice rule that is enabled at $f(\boldsymbol{u})$ and $f(\boldsymbol{u}) \xrightarrow{\chi} u'$ is derivable by the rule. Regardless whether f is τ-preserving or τ-sensitive, due to condition (4), there are three cases:

(1) $f(\boldsymbol{u}) \xrightarrow{\chi} u'$ is derived by an action rule, where the τ-rules associated with the action rule exist.
(2) $f(\boldsymbol{u}) \xrightarrow{\chi} u'$ is derived by a choice rule which has no associated τ-rule.
(3) $f(\boldsymbol{u}) \xrightarrow{\chi} u'$ is derived by a τ-rule.

The proofs of these cases follow below.

(1) Let $f(\boldsymbol{u}) \xrightarrow{\chi} u'$ be derived by a rule r below

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

using a ground substitution ρ defined by $\rho(\boldsymbol{X}) = \boldsymbol{u}$ and $\rho(Y_{ij}) = u_{ij}$ for all i and j. Thus, $f(\boldsymbol{u}) \xrightarrow{\chi} \rho(E[\boldsymbol{X}, \boldsymbol{Y}])$. Let $\boldsymbol{u}'$ denote the vector of all u_{ij}, so $\rho(\boldsymbol{Y}) = \boldsymbol{u}'$. 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. Note that arguments X_i, for $i \in I$, are active in r and, by Definition 19, $E[\boldsymbol{X}, \boldsymbol{Y}]$ contains only τ-preserving operators.

According to Definition 19, rebo rules may have implicit copies provided that the 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, $K = \{i \mid |J_i| > 1\}$ and $|J_i| = 1$ 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 $u' = E[\boldsymbol{u}^\dagger, \boldsymbol{u}']$.

Condition (8) requires that $\{\tau_k \mid k \in K\} \cup \{\tau_l \mid l \in L\} \subseteq \mathit{higher}(r)$, and hence $K \cup L \subseteq \mathit{active}(\mathit{higher}(r))$.

Definition 14 tells us that since r is enabled none of the rules in $\mathit{higher}(r)$ is applicable. Depending on whether f is τ-preserving or τ-sensitive, (6) ensures that all silent rules

that are associated with the rules in $higher(r)$ are members of $higher(r)$. This implies $\rho(X_j) \stackrel{\tau}{\not\rightarrow}$, thus $u_j \stackrel{\tau}{\not\rightarrow}$ for all $j \in active(higher(r))$.

Now, we can use the inductive hypothesis for the pairs of relevant elements in $\boldsymbol{u}$ and $\boldsymbol{v}$ related by $\mathcal{R}$. $u_i \stackrel{\tau}{\not\rightarrow}$, for all $i \in active(higher(r))$, implies $v_i \stackrel{\tau}{\not\rightarrow}$. This is because if we assume $v_i \stackrel{\tau}{\rightarrow}$, we would get, by $(R.c)$ with $u_i \mathcal{R} v_i$, $u_i \stackrel{\tau}{\rightarrow}$: this contradicts the earlier assumption. For each $i \in K$ and $j \in J_i$ we have $u_i \stackrel{a_{ij}}{\rightarrow} u_{ij}$. This implies, by property $(R.a)$,

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

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

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

Let $\boldsymbol{v}'$ stand for the sequence $v_1', \ldots, v_n'$ such that $v_k' = v_k$ when $k \notin I \setminus K$. Let $\boldsymbol{v}^\dagger$ stand for the sequence of only those v_i' where $i \in L$ or $i \notin I$. The elements in $\boldsymbol{v}^\dagger$ are in the order of the corresponding elements of $\boldsymbol{u}^\dagger$ defined above. We have $\boldsymbol{u}^\dagger \mathcal{R} \boldsymbol{v}^\dagger$ as $(u_i^\dagger =)\ u_i \mathcal{E} v_i$ $(= v_i^\dagger)$, for all relevant i, hence $\boldsymbol{u}^\dagger \mathcal{E} \boldsymbol{v}^\dagger$. Moreover, by letting $\boldsymbol{v}''$ to be the vector of all v_{ij}, where $\boldsymbol{v}''$ is constructed in the corresponding way to $\boldsymbol{u}'$, we obtain $\boldsymbol{u}' \mathcal{E} \boldsymbol{v}''$. Since $E[\boldsymbol{X}^\dagger, \boldsymbol{Y}]$ contains only τ-preserving operators Proposition 45 tells us that every component of $\boldsymbol{X}^\dagger$ and $\boldsymbol{Y}$ is τ-preserving in $E[\boldsymbol{X}^\dagger, \boldsymbol{Y}]$. Hence, $E[\boldsymbol{u}^\dagger, \boldsymbol{u}'] \mathcal{E} E[\boldsymbol{v}^\dagger, \boldsymbol{v}'']$.

Finally, we only need to show $f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} C[\boldsymbol{v}'] \stackrel{\chi}{\rightarrow} v''$ for some $C[\boldsymbol{V}]$ and v''. In fact we claim that $C[\boldsymbol{V}] = f(\boldsymbol{X})$ and so $C[\boldsymbol{v}'] = f(\boldsymbol{v}')$. 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 above, we have the following claim. A corresponding result, Claim 37, appears in [53].

Claim 47. *If $\boldsymbol{v} \stackrel{\tau}{\Rightarrow} \boldsymbol{v}^\star \stackrel{\tau}{\Rightarrow} \boldsymbol{v}'$ and $r' \in higher(r)$ is not a τ-rule, then r' does not apply to $f(\boldsymbol{v}^\star)$.*

Proof. Assume that $r' \in higher(r)$ is not a τ-rule and that r' applies to $f(\boldsymbol{v}^\star)$. Since $u_k \stackrel{\tau}{\not\rightarrow}$ for all $k \in active(higher(r))$, we deduce $v_k \stackrel{\tau}{\not\rightarrow}$ for all appropriate k by $(R.c)$. So, $v_j^\star = v_j$, for all $j \in active(r')$, and $u_j \mathcal{R} v_j^\star$. Since the premises of r' are valid for $\boldsymbol{v}^\star$ they are also valid for $\boldsymbol{u}$, thus making r' applicable to $f(\boldsymbol{u})$. This contradicts the assumption that r is enabled at $f(\boldsymbol{u})$. □

$f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} f(\boldsymbol{v}')$ follows from $\boldsymbol{v} \stackrel{\tau}{\Rightarrow} \boldsymbol{v}'$ by the appropriate τ-rules. Since $v_i \stackrel{\tau}{\not\rightarrow}$, for $i \in active(higher(r))$, the τ-rules which may need to be applied belong to the set $T = \{\tau_i \mid i \in I \setminus active(higher(r))\}$. Note that Claim 47 guarantees that these rules are not disabled by non-τ-rules in $higher(r)$. Also, they are not disabled by τ-rules in $higher(r)$ as these are not applicable to $f(\boldsymbol{u})$, and thus to any $f(\boldsymbol{v}^\star)$, where $f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} f(\boldsymbol{v}^\star)$. 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 in T are enabled. Suppose that having reached $f(\boldsymbol{v}^\star)$ we wish to use one of the transitions $v_m^\star \stackrel{\tau}{\rightarrow} v_m^{\star\prime}$, for $m \in M \subseteq I \setminus active(higher(r))$, 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 when none of rules in T 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 to $f(\boldsymbol{v}^\star)$. Thus, $r < r'$ by (7). So by the above claim 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 τ_{m_2} is applicable: this contradicts the earlier statement that none of the rules in $\mathit{higher}(r)$ is applicable to $f(\boldsymbol{v}^\star)$.

Finally, $f(\boldsymbol{v}') \xrightarrow{\chi} v''$ 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 $v'_i \xrightarrow{a_{ij}} v_{ij}$ for all i and j. Furthermore, as explained above, none of the rules in $\mathit{higher}(r)$ can be applied to derive transitions of $f(\boldsymbol{v}')$. Lastly, $v'' = E[\boldsymbol{v}^\dagger, \boldsymbol{v}'']$, and so $u' \mathcal{E} v''$.

(2) Assume that $f(\boldsymbol{u}) \xrightarrow{\chi} u'$ is derived by a choice rule r

$$\frac{X_i \xrightarrow{\chi} Y_i}{f(\boldsymbol{X}) \xrightarrow{\chi} Y_i}$$

using a ground substitution ρ defined by $\rho(\boldsymbol{X}) = \boldsymbol{u}$ and $\rho(Y_j) = u'_i$. Thus, $f(\boldsymbol{u}) \xrightarrow{\chi} u'_i$. Hence, the required context $D[\boldsymbol{X}, \boldsymbol{Y}]$ is simply Y_i, $\boldsymbol{u}' = u' = u'_i$ and $\boldsymbol{u}^\dagger$ is the empty sequence. Moreover, the premise of r is valid under ρ in $\rightarrow$, i.e. $u_i \xrightarrow{\chi} u'_i$.

Conditions (2) and (4) require that f is τ-sensitive and the silent choice rule for f and i, namely τ^i, is among the rules for f. Definition 14 tells us that since r is enabled at $f(\boldsymbol{u})$ none of the rules in $\mathit{higher}(r)$ is applicable to $f(\boldsymbol{u})$. Condition (6) tells us that all silent rules which are associated with the rules in $\mathit{higher}(r)$ are members of $\mathit{higher}(r)$. This implies $\rho(X_k) \xrightarrow{\tau}$, thus $u_k \xrightarrow{\tau}$ for all $k \in \mathit{active}(\mathit{higher}(r))$.

By induction hypothesis, $u_i \xrightarrow{\chi} u'_i$ implies $v_i \stackrel{\tau}{\Rightarrow} v'_i \xrightarrow{\chi} v''_i$ and $u'_i \mathcal{E} v''_i$ for some v'_i and v''_i. As in the previous case we show $u_k \xrightarrow{\tau}$ implies $v_k \xrightarrow{\tau}$ for each $k \in \mathit{active}(\mathit{higher}(r))$.

Finally, we derive $f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} C[\boldsymbol{v}'] \xrightarrow{\chi} v''$, for some $C[\boldsymbol{V}]$, $\boldsymbol{v}'$ and v'', from $v_i \stackrel{\tau}{\Rightarrow} v'_i$. If $v_i \stackrel{\tau}{\Rightarrow}$ stands for one or more τ transitions, then τ^i is used. Conditions (5) and (7) guarantee that τ^i is applicable to $f(\boldsymbol{v})$. We derive $f(\boldsymbol{v}) \xrightarrow{\tau} v$, where v is such that $v_i \xrightarrow{\tau} v \stackrel{\tau}{\Rightarrow} v'_i$, and so $C[\boldsymbol{V}]$ is simply a variable. Moreover, $v'_i \xrightarrow{\chi} v''_i$ so $v'' = v''_i$ and $u' \mathcal{E} v''$. If $v_i \xrightarrow{\tau}$, then $C[\boldsymbol{V}]$ is $f(\boldsymbol{X})$, $f(\boldsymbol{v}) \xrightarrow{\chi} v''$, with $v'' = v''_i$, and $u' \mathcal{E} v''$.

(3) Let $f(\boldsymbol{u}) \xrightarrow{\chi} u'$ be derived by a τ-rule r below

$$\frac{X_i \xrightarrow{\tau} Y_i}{f(\boldsymbol{X}) \xrightarrow{\tau} f(\boldsymbol{X}')},$$

where $X'_j = X_j$, for $j \neq i$, and $X'_i = Y_i$. We use a ground substitution ρ defined by $\rho(\boldsymbol{X}) = \boldsymbol{u}$ and $\rho(Y_i) = u'_i$. Thus, $f(\boldsymbol{u}) \xrightarrow{\tau} f(\boldsymbol{u}')$, where $\boldsymbol{u}' = \rho(\boldsymbol{X}')$. The proof of this case is very similar to the proof of case 1 above.

This completes the proof of property $(\mathcal{R}.a)$.

Property $(\mathcal{R}.\mathbf{c})$: The proof of $(\mathcal{R}.a)$ above. The main difference is that we need the following result in order to use the inductive hypothesis.

Claim 48. *Given $f(\boldsymbol{u})$ and $f(\boldsymbol{v})$ with $f(\boldsymbol{u}) \mathcal{R} f(\boldsymbol{v})$ and $f(\boldsymbol{u})\Downarrow$, if a rule r is enabled at $f(\boldsymbol{v})$, then $u_i \Downarrow$ for all $i \in \mathit{active}(r)$.*

Proof. Assume $f(\boldsymbol{u})\Downarrow$ and let r be enabled at $f(\boldsymbol{v})$. We assume $u_i \Uparrow$, for some $i \in \mathit{active}(r)$, and show that it leads to a contradiction. Either f has the ith τ-rule τ_i or it

has not, in which case condition (8) requires that f has the ith silent choice rule τ^i. In each case we consider if the relevant silent rule is enabled at $f(\boldsymbol{u})$ or not.

Suppose f has τ_i. Assume τ_i is enabled at $f(\boldsymbol{u})$. Since $u_i \Uparrow$ there exists an infinite τ-derivation $u_i \xrightarrow{\tau} u_{i1} \xrightarrow{\tau} u_{i2} \xrightarrow{\tau} \ldots$. For this derivation we can construct an infinite derivation from $f(\boldsymbol{u})$ as follows: Since τ_i is enabled at $f(\boldsymbol{u})$ we obtain $f(\boldsymbol{u}) \xrightarrow{\tau} f(\boldsymbol{u}')$ where $u'_i = u_{i1}$ and $u'_j = u_j$ for $j \neq i$. Now, we show that τ_i is enabled at $f(\boldsymbol{u}')$. If there exists some r^* such that $\tau_i < r^*$ and r^* is enabled at $f(\boldsymbol{u}')$, then $i \notin active(r^*)$ by condition (6). If r^* applies to $f(\boldsymbol{u}')$, then r^* applies also to $f(\boldsymbol{u})$ since $\boldsymbol{u}'$ differs from $\boldsymbol{u}$ only in the ith component and $i \notin active(r^*)$. So, r^* being applicable to $f(\boldsymbol{u})$ contradicts the fact that τ_i is enabled at $f(\boldsymbol{u})$. Thus, τ_i is enabled at $f(\boldsymbol{u}')$. Repeating this argument, we derive an infinite τ-derivation from $f(\boldsymbol{u})$, hence $f(\boldsymbol{u}) \Uparrow$.

Next, suppose τ_i is not enabled at $f(\boldsymbol{u})$. There exists a rule r^* such that $\tau_i < r^*$ and r^* is enabled at $f(\boldsymbol{u})$. By condition (4) or (11), $r < r^*$ since $i \in active(r)$. Applying $r < r^*$ to condition (3) or (10), we also have $r < \tau_j$ for $j \in active(r^*)$. It follows that $v_j \xrightarrow{\tau}\!\!\!\!\!\rightarrow$, for $j \in active(r^*)$, since r is enabled at $f(\boldsymbol{v})$. Let R the set of all rules such as r^* above. We deduce $v_j \xrightarrow{\tau}\!\!\!\!\!\rightarrow$ for $j \in active(R)$. Depending on the form of rules in R, there are three easy cases.

1. There is an action rule, say r', among the rules in R. The premises r' are valid for $f(\boldsymbol{u})$, i.e. $u_j \xrightarrow{a_{jk}} u_{jk}$ for all appropriate j and k. Since $v_j \xrightarrow{\tau}\!\!\!\!\!\rightarrow$, for all $j \in active(r')$, we obtain $v_j \xrightarrow{a_{jk}} v_{jk}$ by property $(R.a)$. Hence, r' also applies to $f(\boldsymbol{v})$. This contradicts the facts that r is enabled at $f(\boldsymbol{v})$ and that $r < r^*$.
2. There are no action rules in R but there is a silent choice rule τ^j. As τ^j is enabled at $f(\boldsymbol{u})$ we deduce $f(\boldsymbol{u}) \xrightarrow{\tau} u'_j$ and $u_j \xrightarrow{\tau} u'_j$. Since $u_j \mathcal{R} v_j$ we obtain $v_j \xrightarrow{\tau} v'_j$ by $(R.a)$. This contradicts the earlier $v_j \xrightarrow{\tau}\!\!\!\!\!\rightarrow$.
3. None of the rules in R is an action rule or a silent choice rule, but R contains a τ-rule τ_j. Since τ_j is enabled at $f(\boldsymbol{u})$ we get $u_j \xrightarrow{\tau} u'_j$. As $u_j \mathcal{R} v_j$ we get $v_j \xrightarrow{\tau} v'_j$ by $(R.a)$. This contradicts $v_j \xrightarrow{\tau}\!\!\!\!\!\rightarrow$.

So far in the proof of Claim 48 we have shown that f cannot have the rule τ_i. Thus, f has the ith silent choice rule τ^i by condition (7). If τ^i is enabled at $f(\boldsymbol{u})$, then we clearly have $f(\boldsymbol{u}) \Uparrow$. Otherwise, there is a rule r^* such that $\tau^i < r^*$ and r^* is enabled at $f(\boldsymbol{u})$. By condition (11), $r < r^*$ since $i \in active(r)$. Similarly as in case 1 of this proof, we can show that r^* applies to $f(\boldsymbol{v})$: this contradicts the fact that r is enabled at $f(\boldsymbol{v})$. □

Now, we return to the proof of property $(\mathcal{R}.c)$. The property is proved by considering the three cases as for $(\mathcal{R}.a)$. In each case the above claim is used to show that, since $u_i \Downarrow$, $v_i \xrightarrow{\chi_{ij}} v_{ij}$ implies $u_i \stackrel{\tau}{\Rightarrow} u'_i \xrightarrow{\chi_{ij}} u_{ij}$, for all χ_{ij} and some u'_i and u_{ij}, and $u_{ij} \mathcal{E} v_{ij}$.

Statement 2. We prove that $\mathcal{E}$ is an eager bisimulation by showing that it satisfies the properties of Proposition 46. Assume $p \mathcal{E} q$ and that properties $(E.a')$, $(E.b')$, $(E.c')$, and $(E.a'')$ and $(E.c'')$ hold for all subterms of p and q that are related by $\mathcal{E}$. Also, assume that that properties $(R.a)$ and $(R.c)$ hold for all subterms of p and q that are related by $\mathcal{R}$. If $p \stackrel{\sqsubset}{\sim} q$, then we are done. Else, p and q can be represented as $f(\boldsymbol{u})$ and $f(\boldsymbol{v})$ respectively, for some f and appropriate $\boldsymbol{u}$ and $\boldsymbol{v}$ such that $u_i \mathcal{E} v_i$, for each i where τ_i for f exists,

and $u_i \mathcal{R} v_i$ otherwise. We prove the following stronger three properties $(\mathcal{E}.a')$, $(\mathcal{E}.b')$ and $(\mathcal{E}.c')$ that originate from $(E.a')$, $(E.b')$ and $(E.c')$, respectively.

$$
\begin{array}{ll}
f(\boldsymbol{u}) \xrightarrow{\chi} u' & \text{implies } \exists C[\boldsymbol{V}], D[\boldsymbol{X}, \boldsymbol{Y}], \boldsymbol{u}^\dagger, \boldsymbol{u}', \boldsymbol{v}^\dagger, \boldsymbol{v}', \boldsymbol{v}'', v''.(f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} C[\boldsymbol{v}'] \xrightarrow{\widehat{\chi}} v'' \\
& \text{and } v'' = D[\boldsymbol{v}^\dagger, \boldsymbol{v}''] \text{ and } u' = D[\boldsymbol{u}^\dagger, \boldsymbol{u}'] \text{ and } u' \mathcal{E} v'') \\
f(\boldsymbol{u}) \stackrel{\tau}{\not\rightarrow} & \text{implies } f(\boldsymbol{v}) \Downarrow \\
f(\boldsymbol{v}) \xrightarrow{\chi} v' & \text{and } f(\boldsymbol{u}) \Downarrow \text{ implies } \exists C[\boldsymbol{V}], D[\boldsymbol{X}, \boldsymbol{Y}], \boldsymbol{v}^\dagger, \boldsymbol{v}', \boldsymbol{u}^\dagger, \boldsymbol{u}', \boldsymbol{u}'', u''. \\
& (f(\boldsymbol{u}) \stackrel{\tau}{\Rightarrow} C[\boldsymbol{u}'] \xrightarrow{\widehat{\chi}} u'' \text{ and } u'' = D[\boldsymbol{u}^\dagger, \boldsymbol{u}''] \\
& \text{and } v' = D[\boldsymbol{v}^\dagger, \boldsymbol{v}'] \text{ and } u'' \mathcal{E} v')
\end{array}
$$

as well as the two properties $(\mathcal{E}.a'')$ and $(\mathcal{E}.c'')$ below that arise from $(E.a'')$ and $(E.c'')$, respectively. Recall, that any a below stands either for a visible action or σ.

$$
\begin{array}{ll}
f(\boldsymbol{u}) \stackrel{\tau}{\not\rightarrow} & \text{implies } \forall v'.[f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} v' \text{ implies } \exists \boldsymbol{v'}.\ (v' = f(\boldsymbol{v'}) \text{ and } \boldsymbol{v} \stackrel{\tau}{\Rightarrow} \boldsymbol{v'} \\
& \text{and } f(\boldsymbol{u}) \mathcal{E} f(\boldsymbol{v'}) \text{ and} \\
& (\mathcal{E}.a^\star) \quad \forall a, u'.\ [f(\boldsymbol{u}) \xrightarrow{a} u' \text{ implies } \exists D[\boldsymbol{X}, \boldsymbol{Y}], \boldsymbol{u}^\dagger, \boldsymbol{u'}, \boldsymbol{v}^\dagger, \boldsymbol{v''}, \boldsymbol{v'''}, v'''. \\
& \quad (f(\boldsymbol{v'}) \stackrel{\tau}{\Rightarrow} f(\boldsymbol{v''}) \xrightarrow{a} v''' \text{ and } \boldsymbol{v'} \stackrel{\tau}{\Rightarrow} \boldsymbol{v''} \text{ and } u' = D[\boldsymbol{u}^\dagger, \boldsymbol{u'}] \\
& \quad \text{and } v''' = D[\boldsymbol{v}^\dagger, \boldsymbol{v'''}] \text{ and } u' \mathcal{E} v''')] \text{ and} \\
& (\mathcal{E}.c^\star) \quad \forall a, v''.\ [f(\boldsymbol{v'}) \xrightarrow{a} v'' \text{ implies } \exists D[\boldsymbol{X}, \boldsymbol{Y}], \boldsymbol{u}^\dagger, \boldsymbol{u'}, \boldsymbol{v}^\dagger, \boldsymbol{v''}, u'. \\
& \quad (f(\boldsymbol{u}) \xrightarrow{a} u' \text{ and } v'' = D[\boldsymbol{v}^\dagger, \boldsymbol{v''}] \text{ and } u' = D[\boldsymbol{u}^\dagger, \boldsymbol{u'}] \text{ and } u' \mathcal{E} v'')])] \\
f(\boldsymbol{v}) \stackrel{\tau}{\not\rightarrow} & \text{and } f(\boldsymbol{u}) \Downarrow \text{ implies } \forall u'.[f(\boldsymbol{u}) \stackrel{\tau}{\Rightarrow} u' \text{ implies } \exists \boldsymbol{u'}.\ (u' = f(\boldsymbol{u'}) \\
& \text{and } \boldsymbol{u} \stackrel{\tau}{\Rightarrow} \boldsymbol{u'} \text{ and } f(\boldsymbol{u'}) \mathcal{E} f(\boldsymbol{v}) \text{ and} \\
& (\mathcal{E}.c^\dagger) \quad \forall a, v'.\ [f(\boldsymbol{v}) \xrightarrow{a} v' \text{ implies } \exists D[\boldsymbol{X}, \boldsymbol{Y}], \boldsymbol{v}^\dagger, \boldsymbol{v'}, \boldsymbol{u}^\dagger, \boldsymbol{u''}, \boldsymbol{u'''}, u'''. \\
& \quad (f(\boldsymbol{u'}) \stackrel{\tau}{\Rightarrow} f(\boldsymbol{u''}) \xrightarrow{a} u''' \text{ and } \boldsymbol{u'} \stackrel{\tau}{\Rightarrow} \boldsymbol{u'''} \text{ and } v' = D[\boldsymbol{v}^\dagger, \boldsymbol{v'}] \\
& \quad \text{and } u''' = D[\boldsymbol{u}^\dagger, \boldsymbol{u'''}] \text{ and } u''' \mathcal{E} v')] \text{ and} \\
& (\mathcal{E}.a^\dagger) \quad \forall a, u''.\ [f(\boldsymbol{u'}) \xrightarrow{a} u'' \text{ implies } \exists D[\boldsymbol{X}, \boldsymbol{Y}], \boldsymbol{v}^\dagger, \boldsymbol{v'}, \boldsymbol{u}^\dagger, \boldsymbol{u''}, v'. \\
& \quad (f(\boldsymbol{v}) \xrightarrow{a} v' \text{ and } u'' = D[\boldsymbol{u}^\dagger, \boldsymbol{u''}] \text{ and } v' = D[\boldsymbol{v}^\dagger, \boldsymbol{v'}] \text{ and } u'' \mathcal{E} v')])]
\end{array}
$$

Property $(\mathcal{E}.\mathbf{a}')$: Assume $f(\boldsymbol{u}) \xrightarrow{\chi} u'$. This means that there is an action rule or a τ-rule or a silent choice rule that is enabled at $f(\boldsymbol{u})$ such that $f(\boldsymbol{u}) \xrightarrow{\chi} u'$ is derived by the rule. There are three cases which are exactly the same as for $(\mathcal{R}.a)$. We prove each of them in turn.

(1) Let $f(\boldsymbol{u}) \xrightarrow{\chi} u'$ be derived by a rule r below:

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

using ρ defined by $\rho(\boldsymbol{X}) = \boldsymbol{u}$ and $\rho(Y_{ij}) = u_{ij}$ for all i and j.

The proof begins in the same manner as the proof of case 1 for property $(\mathcal{R}.a)$ of Statement 1. If $i \in K$, where $K = \{i \mid |J_i| > 1\}$, condition (8) requires $r < \tau_i$. Since r is enabled at $f(\boldsymbol{u})$ we deduce $u_i \stackrel{\tau}{\not\rightarrow}$ for all $i \in K$. Thus, by $(E.b')$ and $(E.a'')$ of the inductive hypothesis, there exists v_i' such that $v_i \stackrel{\tau}{\Rightarrow} v_i' \stackrel{\tau}{\not\rightarrow}$ and $u_i \mathcal{E} v_i'$ for all $i \in K$. For each of these

v'_i property $(E.a^\star)$ tells us that $u_i \stackrel{a_{ij}}{\to} u_{ij}$ (in the premises of r) implies $v'_i \stackrel{a_{ij}}{\to} v'_{ij}$ and $u_{ij} \mathcal{E} v'_{ij}$. Hence,

$$v_i \stackrel{\tau}{\Rightarrow} v'_i \stackrel{a_{ij}}{\to} v'_{ij} \text{ and } u_{ij} \mathcal{E} v'_{ij}. \tag{A.1}$$

Remark 49. Note that (A.1) cannot be derived using an inductive hypothesis based only on the properties from Definition 2. We require the more revealing properties $(E.a'')$ and $(E.c'')$ of Proposition 46.

Similarly as in case 1 for $(\mathcal{R}.a)$, for every $i \in I \backslash K$, transition $u_i \stackrel{a_{i1}}{\to} u_{i1}$ implies, by property $(E.a')$,

$$v_i \stackrel{\tau}{\Rightarrow} v'_i \stackrel{a_{i1}}{\to} v'_{i1} \text{ and } u_{i1} \mathcal{E} v'_{i1} \tag{A.2}$$

Let $\boldsymbol{v'}$ stand for the sequence $v'_1, \ldots, v'_n$ such that:

$$v'_i = \begin{cases} v'_i \text{ as in (A.1) and (A.2)} & \text{for } i \in I \\ v'_i \text{ such that } v_i \stackrel{\tau}{\Rightarrow} v'_i \stackrel{\tau}{\not\to} & \text{for } i \in \mathit{active}(\mathit{higher}(r)) \backslash I \\ v_i & \text{otherwise} \end{cases}$$

Clearly, r applies to $f(\boldsymbol{v'})$. Using a similar argument as in the proof for Claim 50 below, we easily show that if r' is enabled at $f(\boldsymbol{v'})$ and $r < r'$, then r' also applies to $f(\boldsymbol{u})$. This would contradict the assumption that r is enabled at $f(\boldsymbol{u})$. Therefore, r is enabled at $f(\boldsymbol{v'})$. Moreover, similarly as in case 1 for $(\mathcal{R}.a)$, $u' = E[\boldsymbol{u}^\dagger, \boldsymbol{u'}]$ and $v'' = E[\boldsymbol{v}^\dagger, \boldsymbol{v''}]$ where $\boldsymbol{u}^\dagger \mathcal{E} \boldsymbol{v}^\dagger$ and $\boldsymbol{u'} \mathcal{E} \boldsymbol{v''}$.

In order to show $f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} C[\boldsymbol{v'}] \stackrel{\chi}{\to} v''$ and $C[\boldsymbol{X}] = f(\boldsymbol{X})$, we need a result corresponding to Claim 47.

Claim 50. *If $\boldsymbol{v} \stackrel{\tau}{\Rightarrow} \boldsymbol{v}^\star \stackrel{\tau}{\Rightarrow} \boldsymbol{v'}$ and $r' \in \mathit{higher}(r)$ is not a τ-rule, then r' does not apply to $f(\boldsymbol{v}^\star)$.*

Proof. Assume $r' \in \mathit{higher}(r)$ is not a τ-rule and r' applies to $f(\boldsymbol{v}^\star)$. This means that the premises of r' are valid for $f(\boldsymbol{v}^\star)$, namely $v_k^\star \stackrel{a_{kl}}{\to} v_{kl}^\star$ for appropriate k and l. Below, we show that r' applies to $f(\boldsymbol{u})$, which contradicts the earlier assumption that r is enabled at $f(\boldsymbol{u})$.

(a) r' is an action rule with all the associated τ-rules. We have $u_k \stackrel{\tau}{\not\to}$ for $k \in \mathit{active}(r')$. Since $u_k \mathcal{E} v_k$ and $v_k \stackrel{\tau}{\Rightarrow} v_k^\star$ we deduce $u_k \stackrel{a_{kl}}{\to} u_{kl}$ by $(E.c^\star)$. Thus, r' applies to $f(\boldsymbol{u})$.

(b) r' is a choice rule. Assume that the premise of r' for $f(\boldsymbol{v}^\star)$ is $v_i^\star \stackrel{a_i}{\to} v_{i1}^\star (= \rho(X_i \stackrel{a_i}{\to} X_{i1}))$. Since X_i is not τ-preserving in $f(\boldsymbol{X})$, by the definition of $\mathcal{E}$, we have $u_i \mathcal{R} v_i$. Hence, since $u_i \stackrel{\tau}{\not\to}$, we obtain $v_i \stackrel{\tau}{\not\to}$ and $v_i (= v_i^\star) \stackrel{a_i}{\to}$. By $(R.c)$, $u_i \stackrel{a_i}{\to}$, hence r' applies to $f(\boldsymbol{u})$.

(c) r' is a silent choice τ^i. As in case (b) above, $u_i \mathcal{R} v_i$. Because $v_i^\star \stackrel{\tau}{\to}$, we deduce $v_i \stackrel{\tau}{\to}$. Hence, $u_i \stackrel{\tau}{\to}$ by $(R.c)$, and so τ^i applies to $f(\boldsymbol{u})$. □

Returning to case 1 of property $(\mathcal{E}.a')$, we only need to derive $f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} f(\boldsymbol{v'})$. Unlike before, namely in case 1 of property $(\mathcal{R}.a)$, we use the τ-rules from a possibly larger set $\{\tau_k \mid k \in I \cup \mathit{active}(\mathit{higher}(r))\}$. This is because now (A.1) holds for v_k. It is easy to show

that no other τ-rules can be enabled during the derivation $f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} f(\boldsymbol{v}')$. We prove 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 $v_m^\star \stackrel{\tau}{\rightarrow} v_m^{\star'}$, 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 (7) depending whether f is τ-preserving of τ-sensitive. So by Claim 50 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 $\tau_{m_1} < \tau_{m_2} < \ldots$ 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 (7) we obtain $\tau_{m_k} < \tau_{m_k}$. This contradicts (5).

(2) The proof is very similar to the proof of case 2 for $(\mathcal{R}.a)$ since, by the definition of $\mathcal{E}$, the corresponding subterms in $f(\boldsymbol{u})$ and $f(\boldsymbol{v})$ that are active in a choice rule for f are related by $\mathcal{R}$.

(3) The proof of this case differs somewhat from the one for case 3 for $(\mathcal{R}.a)$. It may happen that some v_j have τ transitions, where $j \in \mathit{active}(\mathit{higher}(r))$. But, since $u_j \stackrel{\tau}{\not\rightarrow}$, for all those j, we obtain $v_j \Downarrow$ by the inductive hypothesis. Moreover, if $v_j \stackrel{\tau}{\Rightarrow} v_j' \stackrel{\tau}{\not\rightarrow}$, then $u_j \mathcal{E} v_j'$ by $(E.a'')$. Thus, we obtain $\boldsymbol{v'}$ such that $f(\boldsymbol{u'}) \mathcal{E} f(\boldsymbol{v'})$, and we have $f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} f(\boldsymbol{v'})$ in a similar manner as in case 3 of $(\mathcal{R}.a)$.

Property $(\mathcal{E}.\mathbf{c}')$: The proof is similar to the above proof of $(\mathcal{E}.a')$. It uses property $(E.c'')$, with its subproperties $(E.a^\dagger)$ and $(E.c^\dagger)$, instead of $(E.a'')$ and its respective subproperties. Of course, we need a claim corresponding to Claim 50 that deals with the τ-derivatives of $\boldsymbol{u}$ instead of the τ-derivatives of $\boldsymbol{v}$. In addition, in order to use the $(E.c')$ part of the inductive hypothesis we need an appropriate version of Claim 48 for $\mathcal{E}$.

It is worth noting that case 3 of this claim requires more work than case 3 of Claim 48. For a contradiction we assume $u_i \Uparrow$, for all $i \in I \subseteq \mathit{active}(r)$, and $u_i \Downarrow$ for $i \in \mathit{active}(r) \setminus I$. In case 3 we assume that none of the τ-rules τ_i, for $i \in I$, is enabled at $f(\boldsymbol{u})$ because there is a set of rules R disabling them, and R contains only τ-rules. If $\tau_j \in R$ for some $j \in I$, then $f(\boldsymbol{u}) \Uparrow$ and we are done. Otherwise, since rules in R are enabled at $f(\boldsymbol{u})$ we apply them to derive τ-transitions of $f(\boldsymbol{u})$ and its τ-derivatives. If after any such τ-transitions τ_i is enabled, for some $i \in I$, then clearly $f(\boldsymbol{u}) \Uparrow$. Such situation must occur after a finite number of τ-moves as $u_i \Downarrow$ for all $i \in \mathit{active}(r) \setminus I$.

Next, we move to the proofs of properties $(\mathcal{E}.b')$ and $(\mathcal{E}.a'')$. The following result will be useful. A corresponding result, Claim 38, appears in [53].

Claim 51. *If $f(\boldsymbol{u}) \mathcal{E} f(\boldsymbol{v})$, $f(\boldsymbol{u}) \stackrel{\tau}{\not\rightarrow}$ and $f(\boldsymbol{v}) = q_0 \stackrel{\tau}{\rightarrow} q_1 \stackrel{\tau}{\rightarrow} \cdots \stackrel{\tau}{\rightarrow} q_n$, for $n \geqslant 1$, then $q_i \stackrel{\tau}{\rightarrow} q_{i+1}$ is derived by a τ-rule for $0 \leqslant i < n$.*

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

Firstly, we argue that $f(\boldsymbol{v}) \xrightarrow{\tau} q_1$ is derived by a τ-rule. Assume for a contradiction that a non-τ-rule r with the action τ is enabled at $f(\boldsymbol{v})$. This means that the premises of r are valid for $f(\boldsymbol{v})$. Using the following claim, we obtain $u_k \xrightarrow{\tau}$, and hence $u_k\Downarrow$, for all $k \in \mathit{active}(r)$. There are two types of such rules, namely an action rule with the action τ and the silent choice rule. Before we consider these rules, we first show a useful result.

Claim 52. *If $f(\boldsymbol{u})\mathcal{E}f(\boldsymbol{v})$, $f(\boldsymbol{u}) \xrightarrow{\tau}$, $f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} f(\boldsymbol{v}')$ is derivable by τ-rules, and a rule r is enabled at $f(\boldsymbol{v}')$, then $u_k \xrightarrow{\tau}$ for all $k \in \mathit{active}(r)$.*

Proof. Assume for a contradiction that $u_k \xrightarrow{\tau}$ for some k. If τ_k or τ^k is enabled at $f(\boldsymbol{u})$, then $f(\boldsymbol{u}) \xrightarrow{\tau}$: contradiction. Otherwise, there exists a rule $r^\dagger$ such that $\mathit{act}(r^\dagger) \neq \tau$, $\tau_k < r^\dagger$ or $\tau^k < r^\dagger$, and $r^\dagger$ is enabled at $f(\boldsymbol{u})$. Since $k \in \mathit{active}(r')$, $r' < r^\dagger$ by condition (7). Hence, $r' < \tau_i$ or $r' < \tau^i$ for $i \in \mathit{active}(r^\dagger)$. This implies $v'_i \xrightarrow{\tau}$, for all such i, since r' is enabled at $f(\boldsymbol{v}')$. If $r^\dagger$ is an action rule, then the facts $f(\boldsymbol{u}) \xrightarrow{\tau}$ and $r^\dagger$ is enabled at $f(\boldsymbol{u})$ imply that $u_i \xrightarrow{\tau}$ for all $i \in \mathit{active}(r^\dagger)$. Moreover, since $f(\boldsymbol{u})\mathcal{E}f(\boldsymbol{v})$ we deduce $u_i\mathcal{E}v_i$ by the definition of $\mathcal{E}$ for all $i \in \mathit{active}(r^\dagger)$. Now, we shall use properties $(E.a'')$ and $(E.a^\bigstar)$ of the theorem's inductive hypothesis. By $(E.a'')$, we obtain $u_i\mathcal{E}v'_i$ for all $i \in \mathit{active}(r^\dagger)$ noting that each v'_i is stable as shown above. Since $r^\dagger$ applies to $f(\boldsymbol{u})$, it also applies to $f(\boldsymbol{v}')$ by $(E.a^\bigstar)$. If $r^\dagger$ is a choice rule with the ith argument, then $u_i\mathcal{R}v_i$ with $u_i \xrightarrow{\tau}$. Similarly as for the action rule, we deduce, by $(R.c)$, that $r^\dagger$ applies to $f(\boldsymbol{v}')$. In both cases we have shown that $r^\dagger$ applies to $f(\boldsymbol{v}')$. This contradicts the earlier assumption that r' is enabled at $f(\boldsymbol{v}')$, and $r < r^\dagger$. □

We continue with the proof of Claim 51. We have two cases.

(a) An action rule r with $\mathit{act}(r) = \tau$ is enabled at $f(\boldsymbol{v})$. Hence, $v_k \xrightarrow{a_{kl}} v_{kl}$ for all $k \in \mathit{active}(r)$ and appropriate l. Having shown $u_k \xrightarrow{\tau}$, for all $k \in \mathit{active}(r)$, we apply $(E.c')$ to $u_k\mathcal{E}v_k$. From $v_k \xrightarrow{a_{kl}} v_{kl}$ we obtain $u_k \xrightarrow{a_{kl}} u_{kl}$, which implies that r also applies to $f(\boldsymbol{u})$. If r is enabled at $f(\boldsymbol{u})$, then it contradicts the assumption that $f(\boldsymbol{u}) \xrightarrow{\tau}$. Else, there must exist r' such that $r < r'$, $\mathit{act}(r') \neq \tau$ and r' is enabled at $f(\boldsymbol{u})$. This implies $r < \tau_j$ or $r < \tau^j$, and hence $v_j \xrightarrow{\tau}$, for $j \in \mathit{active}(r')$, since r is enabled at $f(\boldsymbol{v}')$. By $(E.a')$, since the premises of r' are valid for $f(\boldsymbol{u})$ they are also valid for $f(\boldsymbol{v})$, making r' applicable to $f(\boldsymbol{v})$: contradiction.

(b) A silent choice rule τ^k is enabled at $f(\boldsymbol{v})$. Hence, $v_k \xrightarrow{\tau}$. Since the kth argument of f is not τ-preserving in $f(\boldsymbol{X})$, by the definition of $\mathcal{E}$, we have $u_k\mathcal{R}v_k$. Using $(R.c)$, since $u_k\Downarrow$ we obtain $u_k \xrightarrow{\tau}$ from $v_k \xrightarrow{\tau}$. This contradicts the earlier deduction that $u_k \xrightarrow{\tau}$.

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 $0 \leqslant i < k$, where $k < n$, and prove that $q_k \xrightarrow{\tau} q_{k+1}$ is derived by a τ-rule. Our assumption means that $q_0 \stackrel{\tau}{\Rightarrow} q_k$ and $q_k = f(\boldsymbol{v}')$ for some $\boldsymbol{v}'$ such that $\boldsymbol{v} \stackrel{\tau}{\Rightarrow} \boldsymbol{v}'$. 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 $\mathit{act}(r') = \tau$ is enabled at $f(\boldsymbol{v}')$. r' is either an action rule with the action τ or the silent choice rule. The premises of r' are valid for $f(\boldsymbol{v}')$: $v'_k \xrightarrow{a_{kl}} v'_{kl}$ for all $k \in \mathit{active}(r')$ and appropriate l. By Claim 52, we have $u_k \xrightarrow{\tau}$ and $u_k\Downarrow$, for all $k \in \mathit{active}(r')$. Hence, $\boldsymbol{u}\mathcal{E}\boldsymbol{v}'$ by $(E.c')$.

Remark 53. Notice that at this point, although we have $\boldsymbol{u}\mathcal{E}\boldsymbol{v}'$, we cannot use $(E.c')$ and $(E.a')$ for the pairs of related processes from $\boldsymbol{u}$ and $\boldsymbol{v}'$ to proceed with the proof. The inductive hypothesis guarantees that only those components of $\boldsymbol{u}$ and $\boldsymbol{v}$ that are related by $\mathcal{E}$ satisfy properties $(E.a')$ and $(E.c')$. Their τ-derivatives, which may have grown in size, are not assumed to have these properties. Their behaviour, however, can be deduced from the properties $(E.c'')$ and $(E.a'')$ of the inductive hypothesis. This is the main reason for including these two more revealing properties in the characterisation of eager bisimulation.

Returning to the proof of Claim 51, we consider two cases.

(a) An action rule r' is enabled at $f(\boldsymbol{v}')$. Since $v'_k \stackrel{a_{kl}}{\rightarrow} v'_{kl}$ for all $k \in \mathit{active}(r')$ and appropriate l, and $u_k \stackrel{\tau}{\not\rightarrow}$, for all $k \in \mathit{active}(r')$, we apply $(E.c^{\bigstar})$ to $u_k \mathcal{E} v'_k$. From $v'_k \stackrel{a_{kl}}{\rightarrow} v'_{kl}$ we obtain $u_k \stackrel{a_{kl}}{\rightarrow} u_{kl}$, which implies that r' also applies to $f(\boldsymbol{u})$. If r' is enabled at $f(\boldsymbol{u})$, then it contradicts the assumption that $f(\boldsymbol{u}) \stackrel{\tau}{\not\rightarrow}$. Else, there must exist r'' such that $r' \prec r''$, $\mathit{act}(r'') \neq \tau$ and r'' is enabled at $f(\boldsymbol{u})$. This implies $r' \prec \tau_j$ or $r' \prec \tau^j$, and hence $v'_j \stackrel{\tau}{\not\rightarrow}$, for $j \in \mathit{active}(r'')$, as r' is enabled at $f(\boldsymbol{v}')$. By $(E.a^{\bigstar})$, since the premises of r'' are valid for $f(\boldsymbol{u})$ they are also valid for $f(\boldsymbol{v}')$, making r'' applicable to $f(\boldsymbol{v}')$: contradiction.

(b) A silent choice rule τ^k is enabled at $f(\boldsymbol{v}')$. This case is proved in the same fashion as case (b) above.

By Claim 51 all τ-derivatives of $f(\boldsymbol{v})$ have the form $f(\boldsymbol{v}')$, where $\boldsymbol{v} \stackrel{\tau}{\Rightarrow} \boldsymbol{v}'$. □

Property $(\mathscr{E}.\mathbf{b}')$: Suppose for a contradiction that $f(\boldsymbol{v})\Uparrow$. Claim 51 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 $f(\boldsymbol{v})\Uparrow$ implies $v_m\Uparrow$ for some $m \in M$. If $u_m \stackrel{\tau}{\not\rightarrow}$, then $v_m\Uparrow$ contradicts the $(E.b')$ part of the inductive hypothesis. If $u_m \stackrel{\tau}{\rightarrow}$, then since $f(\boldsymbol{u}) \stackrel{\tau}{\not\rightarrow}$ the set of rules other than τ_m 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, $v_k^{\dagger} \stackrel{\tau}{\not\rightarrow}$ 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, for all appropriate k, $v_k^{\dagger} \stackrel{\tau}{\not\rightarrow}$ and by the $(E.a'')$ 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.

Property $(\mathscr{E}.\mathbf{a}'')$: The above claim tells us that all τ-derivatives of $f(\boldsymbol{v})$ have the form $f(\boldsymbol{v}')$, where $\boldsymbol{v} \stackrel{\tau}{\Rightarrow} \boldsymbol{v}'$. We show $f(\boldsymbol{u})\mathcal{E}f(\boldsymbol{v}')$ as follows: Assume that in the process of deriving $f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} f(\boldsymbol{v}')$ we only used τ-rules from the set $\{\tau_m \mid m \in M\}$. We easily see, by Claim 52, that $u_m \stackrel{\tau}{\not\rightarrow}$ for all $m \in M$. This gives us $u_m \mathcal{E} v'_m$, for $m \in M$, by property $(E.a'')$. For $i \notin M$ we have $v'_i = v_i$, so $u_i \mathcal{E} v'_i$ or $u_i \mathcal{R} v'_i$. Therefore, $f(\boldsymbol{u})\mathcal{E}f(\boldsymbol{v}')$ by the definition of $\mathcal{E}$.

Subproperty $(\mathscr{E}.\mathbf{a}^{\bigstar})$: There are two types of rules that $f(\boldsymbol{u}) \stackrel{a}{\rightarrow} u'$ can be derived by. Firstly, we have an action rule with all the associated τ-rules. Let this rule be r as in case 1 of property $(\mathcal{E}.a')$, Statement 2. The proof follows along the same lines as in the mentioned case. Secondly, we consider a choice rule. The proof here is a simpler version of the proof for case 2 for $(\mathcal{E}.a')$.

Subproperty ($\mathscr{E}.\mathbf{c}^{\star}$): Assume that $f(\boldsymbol{v'}) \stackrel{a}{\rightarrow} v''$ is derived by r as in case 1 of $(\mathcal{E}.a')$. The premises of r are valid for $f(\boldsymbol{v'})$, namely $v'_i \stackrel{a_{ij}}{\rightarrow} v'_{ij}$ for all appropriate i and j. By $(E.c^{\star})$ of the inductive hypothesis, for all relevant pairs of u_i and v_i, we obtain $u_i \stackrel{a_{ij}}{\rightarrow} u_{ij}$. Hence, r applies to $f(\boldsymbol{u})$. We need to show that no rule higher than r applies to $f(\boldsymbol{u})$. If we assume that there exists r' such that $r < r'$ and r' applies to $f(\boldsymbol{u})$, we easily find, using the inductive hypothesis, that r' also applies to $f(\boldsymbol{v})$. Here, we use the fact that $u_k \stackrel{\tau}{\not\rightarrow}$ and $v_k \stackrel{\tau}{\not\rightarrow}$ for all $k \in active(r) \cup active(r')$. Hence, r is enabled at $f(\boldsymbol{u})$, and by applying r to $f(\boldsymbol{u})$ we get the required u' where $f(\boldsymbol{u}) \stackrel{a}{\rightarrow} u'$. We show $u' \mathcal{E} v''$ as before by using the definition of $\mathcal{E}$.

Property ($\mathscr{E}.\mathbf{c}''$): Since it is very similar to $(\mathcal{E}.a'')$, with $f(\boldsymbol{u})$ and $f(\boldsymbol{v})$ "almost" swapping places, its proof follows closely the above proof of $(\mathcal{E}.a'')$. Naturally, we need a version of Claim 51 which states that if $f(\boldsymbol{v})$ is stable and $f(\boldsymbol{u})$ is convergent, then τ-derivatives of $f(\boldsymbol{u})$ can only be produced by τ-rules.

B. Congruence theorem for timed rooted eager bisimulation

Let $G = (\Sigma, \mathsf{Act}, R, <)$ be a timed process language and let $\rightarrow$ be the transition relation generated by G as in Definition 13. We start by defining relations $\mathcal{R}_\sigma$ and $\mathcal{E}_\sigma$ over $T(\Sigma) \times T(\Sigma)$ as in the proof of Theorem 22. $\mathcal{R}_\sigma$ is the least relation over $T(\Sigma) \times T(\Sigma)$ satisfying the following two conditions:

- $\boldsymbol{u} \mathcal{R}_\sigma \boldsymbol{v}$ if $\boldsymbol{u} \sqsubseteq_{tr} \boldsymbol{v}$, and
- $C[\boldsymbol{u}] \mathcal{R}_\sigma C[\boldsymbol{v}]$ if $\boldsymbol{u} \mathcal{R}_\sigma \boldsymbol{v}$,

where $C[\boldsymbol{X}]$ is any Σ context and $\boldsymbol{u}$ and $\boldsymbol{v}$ are vectors of the right length of closed Σ terms. $\mathcal{E}_\sigma$ is the least relation satisfying the following two conditions:

- $\boldsymbol{u} \mathcal{E}_\sigma \boldsymbol{v}$ if $\boldsymbol{u} \sqsubseteq \boldsymbol{v}$ and,
- $C[\boldsymbol{u}] \mathcal{E}_\sigma C[\boldsymbol{v}]$ if $u_i \mathcal{E}_\sigma v_i$, whenever X_i is τ-preserving in $C[\boldsymbol{X}]$, and $u_i \mathcal{R}_\sigma v_i$, otherwise, for each i.

As before we need to show that $\mathcal{R}_\sigma$ is a timed rooted eager bisimulation relation and that $\mathcal{E}_\sigma$ is an eager bisimulation relation. We prove in parallel the following two statements by induction of the depth of process terms.

Statement 1. If $p \mathcal{R}_\sigma q$, then p and q satisfy $(R_\sigma.a1)$, $(R_\sigma.a2)$, $(R_\sigma.c1)$ and $(R_\sigma.c2)$ in Definition 32.

Statement 2. If $p \mathcal{E}_\sigma q$, then p and q satisfy $(E.a')$, $(E.b')$, $(E.c')$, $(E.a'')$ and $(E.c'')$ in Proposition 46.

Statement 1. Assume $p \mathcal{R} q$ and that $(R_\sigma.a1)$, $(R_\sigma.a2)$, $(R_\sigma.c1)$ and $(R_\sigma.c2)$ hold for all subterms of p and q that are related by $\mathcal{R}_\sigma$. If $p \sqsubseteq_{tr} q$, then we are done. Else, p and q can be represented as $f(\boldsymbol{u})$ and $f(\boldsymbol{v})$ respectively for some f, $\boldsymbol{u}$ and $\boldsymbol{v}$ with $\boldsymbol{u} \mathcal{R} \boldsymbol{v}$. Correspondingly as before, we show that the following stronger properties, denoted by $(\mathcal{R}_\sigma.a1)$, $(\mathcal{R}_\sigma.a2)$, $(\mathcal{R}_\sigma.c1)$ and $(\mathcal{R}_\sigma.c2)$ respectively, hold.

$f(\boldsymbol{u}) \xrightarrow{\alpha} u'$ implies $\exists C[V], D[X, Y], \boldsymbol{u}^\dagger, \boldsymbol{u}', \boldsymbol{v}^\dagger, \boldsymbol{v}', \boldsymbol{v}'', v''$.
$(f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} C[\boldsymbol{v}'] \xrightarrow{\alpha} v''$ and $v'' = D[\boldsymbol{v}^\dagger, \boldsymbol{v}'']$
and $u' = D[\boldsymbol{u}^\dagger, \boldsymbol{u}']$ and $u' \mathcal{E}_\sigma v'')$

$f(\boldsymbol{u}) \xrightarrow{\sigma} u'$ implies $\exists D[X, Y], \boldsymbol{u}', \boldsymbol{v}', v'.(f(\boldsymbol{v}) \xrightarrow{\sigma} v'$
and $v' = D[\boldsymbol{v}, \boldsymbol{v}']$ and $u' = D[\boldsymbol{u}, \boldsymbol{u}']$ and $u' \mathcal{R}_\sigma v')$

$f(\boldsymbol{v}) \xrightarrow{\alpha} v'$ and $f(\boldsymbol{u})\Downarrow$ implies $\exists C[V], D[X, Y], \boldsymbol{v}^\dagger, \boldsymbol{v}', \boldsymbol{u}^\dagger, \boldsymbol{u}', \boldsymbol{u}'', u''$.
$(f(\boldsymbol{u}) \stackrel{\tau}{\Rightarrow} C[\boldsymbol{u}'] \xrightarrow{\alpha} u''$ and $u'' = D[\boldsymbol{u}^\dagger, \boldsymbol{u}'']$
and $v' = D[\boldsymbol{v}^\dagger, \boldsymbol{v}']$ and $u'' \mathcal{E}_\sigma v')$

$f(\boldsymbol{v}) \xrightarrow{\sigma} v'$ and $f(\boldsymbol{u})\Downarrow$ implies $\exists D[X, Y], \boldsymbol{u}', \boldsymbol{v}', u'.(f(\boldsymbol{u}) \xrightarrow{\sigma} u'$
and $v' = D[\boldsymbol{v}, \boldsymbol{v}']$ and $u' = D[\boldsymbol{u}, \boldsymbol{u}']$ and $u' \mathcal{R}_\sigma v')$

Properties $(\mathcal{R}_\sigma.a1)$ and $(\mathcal{R}_\sigma.c1)$ are proved in a similar way as the corresponding properties $(\mathcal{R}.a)$ and $(\mathcal{R}.c)$ in the previous section.

Property $(\boldsymbol{\mathcal{R}_\sigma}$**.a2**): f is either time altering or time preserving. Firstly, we consider a simpler case, namely f is time preserving. Then, $f(\boldsymbol{u}) \xrightarrow{\sigma} u'$ is by a σ-rule σ_I. Hence, $u' = f(\boldsymbol{u'})$ and $D[X, Y] = f(Y)$ where $u_i \xrightarrow{\sigma} u_i'$, for $i \in I$, and $u_i = u_i'$ for $i \notin I$. By $(R.a2)$ of the inductive hypothesis since $\boldsymbol{u} \mathcal{R}_\sigma \boldsymbol{v}$ we obtain $v_i \xrightarrow{\sigma} v_i'$, for all $i \in I$, and $\boldsymbol{u'} \mathcal{R}_\sigma \boldsymbol{v'}$. So, σ_I also applies to $f(\boldsymbol{v})$. If σ_I is enabled at $f(\boldsymbol{v})$, then $f(\boldsymbol{v}) \xrightarrow{\sigma} f(\boldsymbol{v'})$, and $f(\boldsymbol{u'}) \mathcal{R}_\sigma f(\boldsymbol{v'})$ as $\boldsymbol{u'} \mathcal{R}_\sigma \boldsymbol{v'}$.

If σ_I is not enabled at $f(\boldsymbol{v})$, then there is a rule $r \in \mathit{higher}(\sigma_I)$ that applies to $f(\boldsymbol{v})$. Since no rules other than σ-rules can be above any timed rules for time preserving f, $r = \sigma_J$. Since $\sigma_I < \sigma_J$ we deduce $\sigma_I < \tau(j)$ for all $j \in J$, by condition (6). As σ_I is enabled at $f(\boldsymbol{u})$ we get $u_j \stackrel{\tau}{\not\rightarrow}$, for all $j \in J$, hence $u_j \Downarrow$ for all appropriate j. By $(R.c2)$ of the inductive hypothesis since σ_J applies to $f(\boldsymbol{v})$ we deduce that σ_J applies also to $f(\boldsymbol{u})$: this contradicts to the assumption that σ_I is enabled at $f(\boldsymbol{u})$. Hence, σ_I is enabled at $f(\boldsymbol{v})$, and we are done.

Secondly, assume that f is time altering, and $f(\boldsymbol{u}) \xrightarrow{\sigma} E[\boldsymbol{u}, \boldsymbol{u'}]$ by a timed rule r_σ where $u_i \xrightarrow{\sigma} u_i'$ for $i \in \mathit{active}(r)$. Similarly to the previous case, r_σ applies to $f(\boldsymbol{v})$. Assume for a contradiction that there exists $r \in \mathit{higher}(r_\sigma)$ that applies to $f(\boldsymbol{v})$. As f is time altering we have three cases.

1. Let r be a timed rule. By a corresponding argument as above, all the premises of r are also valid for $f(\boldsymbol{u})$. This contradicts the assumption that r_σ is enabled at $f(\boldsymbol{u})$.
2. Let r be an action rule. All the premises of r are valid for $f(\boldsymbol{v})$. Since $r_\sigma < r$ we deduce $r_\sigma < \tau(j)$ for all $j \in \mathit{active}(r)$ by condition (6). As r_σ is enabled at $f(\boldsymbol{u})$ we get $u_j \stackrel{\tau}{\not\rightarrow}$ and, hence, $u_j \Downarrow$ for all appropriate j. Now, by $(R_\sigma.c1)$, we deduce that the premises of r are also valid for $f(\boldsymbol{u})$, hence r applies to $f(\boldsymbol{u})$: contradiction.
3. Let r be $\mathit{tau}(k)$. Since $\mathit{tau}(k)$ applies to $f(\boldsymbol{v})$ we get $v_k \xrightarrow{\tau}$. As in case 2 above we deduce $u_k \stackrel{\tau}{\not\rightarrow}$. This contradicts the inductive assumption that $(R_\sigma.a1)$ holds for u_k and v_k.

Hence, r_σ is enabled at $f(\boldsymbol{v})$ and $f(\boldsymbol{v}) \xrightarrow{\sigma} E[\boldsymbol{v}, \boldsymbol{v'}]$, with $\boldsymbol{u'} \mathcal{R}_\sigma \boldsymbol{v'}$. Finally, $E[\boldsymbol{u}, \boldsymbol{u'}] \mathcal{R}_\sigma E[\boldsymbol{v}, \boldsymbol{v'}]$ as $\boldsymbol{u} \mathcal{R}_\sigma \boldsymbol{v}$ and $\boldsymbol{u'} \mathcal{R}_\sigma \boldsymbol{v'}$.

Property ($\mathscr{R}_\sigma$**.c2**): It is proved using an auxiliary result corresponding to Claim 48 when r is a timed rule. We consider two cases when f is time preserving and then f is time altering, and argue as for property $(\mathcal{R}_\sigma.a2)$ above.

Statement 2. Next, we prove $\mathcal{E}_\sigma$ is an eager bisimulation by showing that the properties of Proposition 46 hold. Assume $p\mathcal{E}_\sigma q$ and that properties $(E.a')$, $(E.b')$, $(E.c')$, and $(E.a'')$ and $(E.c'')$ hold for all subterms of p and q that are related by $\mathcal{E}_\sigma$. Also, assume that properties $(R_\sigma.a1)$, $(R_\sigma.a2)$, $(R_\sigma.c1)$ and $(R_\sigma.c2)$ hold for all subterms of p and q that are related by $\mathcal{R}_\sigma$. If $p \not\simeq q$, then we are done. Else, p and q can be represented as $f(\boldsymbol{u})$ and $f(\boldsymbol{v})$ respectively, for some f and appropriate $\boldsymbol{u}$ and $\boldsymbol{v}$ such that $u_i\,\mathcal{E}_\sigma\,v_i$, for each $i \in \mathit{active}(f)$ where τ_i for f exists, and $u_i\,\mathcal{R}_\sigma\,v_i$ otherwise (i.e. when τ^i exists by (2)).

As in the congruence proof in the previous section we show five stronger properties instead of the properties in Proposition 46. The proofs of the five properties for visible and silent actions are simpler versions of the corresponding proofs in Statement 2 in the previous section, and thus omitted. This is because the rules with visible and silent actions and orderings on such rules in timed rebo process languages satisfy stricter constraints than the corresponding rules and orderings in rebo process languages. For example, we cannot have rules with visible and σ actions in timed rebo process languages but they are allowed in rebo process languages. The corresponding statement does not hold for timed rules in general: we can have τ-sensitive operators in the targets on timed rules (for time preserving operators) and rules τ-sensitive operators in the targets are not permitted in rebo process languages. Thus, we only prove the following four properties with action σ. We denote them by $(\mathcal{E}_\sigma.a'_\sigma)$, $(\mathcal{E}_\sigma.c'_\sigma)$, $(\mathcal{E}_\sigma.a''_\sigma)$ and $(\mathcal{E}_\sigma.c''_\sigma)$, respectively.

$f(\boldsymbol{u}) \xrightarrow{\sigma} u'$ implies $\exists D[\boldsymbol{X}, \boldsymbol{Y}], \boldsymbol{u}^\dagger, \boldsymbol{u}', \boldsymbol{v}^\dagger, \boldsymbol{v}', \boldsymbol{v}'', v''.(f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} f(\boldsymbol{v}') \xrightarrow{\sigma} v''$
and $v'' = D[\boldsymbol{v}^\dagger, \boldsymbol{v}'']$ and $u' = D[\boldsymbol{u}^\dagger, \boldsymbol{u}']$ and $u'\mathcal{E}_\sigma v'')$

$f(\boldsymbol{v}) \xrightarrow{\sigma} v'$ and $f(\boldsymbol{u})\Downarrow$ implies $\exists D[\boldsymbol{X}, \boldsymbol{Y}], \boldsymbol{v}^\dagger, \boldsymbol{v}', \boldsymbol{u}^\dagger, \boldsymbol{u}', \boldsymbol{u}'', u''.$
$(f(\boldsymbol{u}) \stackrel{\tau}{\Rightarrow} f(\boldsymbol{u}') \xrightarrow{\sigma} u''$ and $u'' = D[\boldsymbol{u}^\dagger, \boldsymbol{u}'']$
and $v' = D[\boldsymbol{v}^\dagger, \boldsymbol{v}']$ and $u''\mathcal{E}_\sigma v')$

$f(\boldsymbol{u}) \stackrel{\tau}{\not\to}$ implies $\forall q'.[f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} q'$ implies $\exists \boldsymbol{v'}.$ $(q' = f(\boldsymbol{v'})$ and $\boldsymbol{v} \stackrel{\tau}{\Rightarrow} \boldsymbol{v'}$
and $f(\boldsymbol{u})\mathcal{E}_\sigma f(\boldsymbol{v'})$ and
$(\mathcal{E}_\sigma.a^\star_\sigma)$ $\forall u'.[f(\boldsymbol{u}) \xrightarrow{\sigma} u'$ implies $\exists D[\boldsymbol{X}, \boldsymbol{Y}], \boldsymbol{u}^\dagger, \boldsymbol{u'}, \boldsymbol{v}^\dagger, \boldsymbol{v''}, \boldsymbol{v'''}, v'''.$
$(f(\boldsymbol{v'}) \stackrel{\tau}{\Rightarrow} f(\boldsymbol{v''}) \xrightarrow{\sigma} v''$ and $\boldsymbol{v'} \stackrel{\tau}{\Rightarrow} \boldsymbol{v''}$ and $u' = D[\boldsymbol{u}^\dagger, \boldsymbol{u'}]$
and $v'' = D[\boldsymbol{v}^\dagger, \boldsymbol{v''}]$ and $u'\mathcal{E}_\sigma v'')]$ and
$(\mathcal{E}_\sigma.c^\star_\sigma)$ $\forall v''.$ $[f(\boldsymbol{v'}) \xrightarrow{\sigma} v''$ implies $\exists D[\boldsymbol{X}, \boldsymbol{Y}], \boldsymbol{u}^\dagger, \boldsymbol{u'}, \boldsymbol{v}^\dagger, \boldsymbol{v''}, u'.$
$(f(\boldsymbol{u}) \xrightarrow{\sigma} u'$ and $v'' = D[\boldsymbol{v}^\dagger, \boldsymbol{v''}]$ and $u' = D[\boldsymbol{u}^\dagger, \boldsymbol{u'}]$ and $u'\mathcal{E}_\sigma v'')])]$

$f(\boldsymbol{v}) \stackrel{\tau}{\not\to}$ and $f(\boldsymbol{u})\Downarrow$ implies $\forall p'.[f(\boldsymbol{u}) \stackrel{\tau}{\Rightarrow} p'$ implies $\exists \boldsymbol{u'}.$ $(p' = f(\boldsymbol{u'})$
and $\boldsymbol{u} \stackrel{\tau}{\Rightarrow} \boldsymbol{u'}$ and $f(\boldsymbol{u'})\mathcal{E}_\sigma f(\boldsymbol{v})$ and
$(\mathcal{E}.c^\dagger_\sigma)$ $\forall v'.$ $[f(\boldsymbol{v}) \xrightarrow{\sigma} v'$ implies $\exists D[\boldsymbol{X}, \boldsymbol{Y}], \boldsymbol{v}^\dagger, \boldsymbol{v'}, \boldsymbol{u}^\dagger, \boldsymbol{u''}, \boldsymbol{u'''}, u'''.$
$(f(\boldsymbol{u'}) \stackrel{\tau}{\Rightarrow} f(\boldsymbol{u''}) \xrightarrow{\sigma} u''$and $\boldsymbol{u'} \stackrel{\tau}{\Rightarrow} \boldsymbol{u''}$ and $v' = D[\boldsymbol{v}^\dagger, \boldsymbol{v'}]$
and $u''' = D[\boldsymbol{u}^\dagger, \boldsymbol{u'''}]$ and $u''\mathcal{E}_\sigma v')]$ and
$(\mathcal{E}.a^\dagger_\sigma)$ $\forall u''.$ $[f(\boldsymbol{u'}) \xrightarrow{\sigma} u''$ implies $\exists D[\boldsymbol{X}, \boldsymbol{Y}], \boldsymbol{v}^\dagger, \boldsymbol{v'}, \boldsymbol{u}^\dagger, \boldsymbol{u''}, v'.$
$(f(\boldsymbol{u}) \xrightarrow{\sigma} u'$ and $u' = D[\boldsymbol{v}^\dagger, \boldsymbol{u'}]$ and $v' = D[\boldsymbol{u}^\dagger, \boldsymbol{v'}]$ and $u''\mathcal{E}_\sigma v')])].$

Property $(\mathscr{E}_\sigma.a_\sigma)$: There are three cases:

(1) f is time preserving and τ-preserving and $f(\boldsymbol{u}) \stackrel{\sigma}{\to} u'$ is derived by σ_I.

(2) f is time preserving and τ-sensitive and $f(\boldsymbol{u}) \stackrel{\sigma}{\to} u'$ is derived by σ_I.

(3) f is time altering and $f(\boldsymbol{u}) \stackrel{\sigma}{\to} u'$ is derived by a timed rule r_σ.

We show that the property holds for each of these cases.

(1) The form of σ_I implies that $u' = f(\boldsymbol{u'})$ and $u_i \stackrel{\sigma}{\to} u'_i$ for $i \in I$ and $u_i = u'_i$ for $i \notin I$. As σ_I has no implicit copies the proof of this case is a simpler version of the proof for case 1 of property $(\mathcal{E}.a')$ for Theorem 22.

(2) The proof is similar to a large extend to that of case 1 of $(\mathcal{E}.a')$. However, since f is τ-sensitive it may happen that some of the active arguments in σ_I are not τ-preserving in $f(\boldsymbol{X})$ since f may have silent choice rules (and not the τ-rules) for those arguments. An example of such f is the time preserving version of CCS $+$. Hence, the proof of the statement $u' \mathcal{E}_\sigma v''$ is similar to the proofs of cases 2 for $(\mathcal{E}.a')$ and $(\mathcal{R}.a)$. Note that Claim 50 is valid for $r = \sigma_I$, so only τ-rules are used in the derivation of $f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} f(\boldsymbol{v'})$. Then, $f(\boldsymbol{v'}) \stackrel{\sigma}{\to} v''$ by rule σ_I. Because of the form of σ_I, we deduce $u' = f(\boldsymbol{u'})$ and $v'' = f(\boldsymbol{v''})$ for the appropriate vectors $\boldsymbol{u'}$ and $\boldsymbol{v''}$. By the inductive hypothesis we have $u'_i \mathcal{E}_\sigma v''_i$ for all i such that X_i is τ-preserving in $f(\boldsymbol{X})$, and $u'_i \mathcal{R}_\sigma v''_i$ otherwise. This implies $f(\boldsymbol{u'}) \mathcal{E}_\sigma f(\boldsymbol{v''})$ by the definition of $\mathcal{E}_\sigma$.

(3) Let r_σ be the following rule. As in case 1 of property $(\mathcal{E}.a')$ we use r_σ to show that $(\mathcal{E}_\sigma.a_\sigma)$ holds.

$$\frac{\{X_i \stackrel{\sigma}{\to} X'_i\}_{i \in I}}{f(\boldsymbol{X}) \stackrel{\sigma}{\to} D[\boldsymbol{X}, \boldsymbol{X'}]}.$$

Although $D[\boldsymbol{X}, \boldsymbol{X'}]$, the target of r_σ, may be more complex than the target of a typical σ-rule such as σ_I above, it contains only τ-preserving operators by Definition 33. Proposition 45 says that all variables in $D[\boldsymbol{X}, \boldsymbol{X'}]$ are τ-preserving. Hence, we do not need to be concerned, as in case 2, that some processes related by $\mathcal{E}_\sigma$ may end up in non-τ-preserving places in $D[\boldsymbol{X}, \boldsymbol{X'}]$. So, $D[\boldsymbol{u}, \boldsymbol{u'}] \mathcal{E}_\sigma D[\boldsymbol{v}, \boldsymbol{v''}]$ holds because the pairs of vectors of relevant processes, for example $\boldsymbol{u'}$ and $\boldsymbol{v''}$, are related by either $\mathcal{E}_\sigma$ or $\mathcal{R}_\sigma$, and we can apply properties $(E.a')$ and $(R.a)$ of the inductive hypothesis.

Property $(\mathscr{E}_\sigma.\mathbf{c}'_\sigma)$: The proof of this property relates to the proof of $(\mathcal{E}_\sigma.a'_\sigma)$ in the same way as the proof of $(\mathcal{E}.c')$ related to $(\mathcal{E}.a')$ are related. Of course the cases that need to be considered are somewhat different but the approach and the proof 'tricks' are very similar. We use property $(E.c'')$ with action a being σ, and its subproperties $(E.a^\dagger)$ and $(E.c^\dagger)$, instead of $(E.a'')$ and its respective subproperties. In order to use the $(E.c')$ part of the inductive hypothesis, where the action a is σ, we need an appropriate version of Claim 48 for $\mathcal{E}_\sigma$.

Property $(\mathscr{E}_\sigma.\mathbf{a}''_\sigma)$: A claim corresponding to Claim 51 tells us that all τ-derivatives of $f(\boldsymbol{v})$ have the form $f(\boldsymbol{v'})$, where $\boldsymbol{v} \stackrel{\tau}{\Rightarrow} \boldsymbol{v'}$. We show $f(\boldsymbol{u}) \mathcal{E} f(\boldsymbol{v'})$ in the same manner as for $(\mathcal{E}.a'')$ in the previous section.

Next, we consider $(\mathcal{E}_\sigma.a^\star_\sigma)$ and $(\mathcal{E}_\sigma.c^\star_\sigma)$. The structure of the proofs is similar to that of the proofs for $(\mathcal{E}.a^\star)$ and $(\mathcal{E}.c^\star)$ in the previous section. We use the same three cases as for $(\mathcal{E}_\sigma.a'_\sigma)$ above.

Property $(\mathscr{E}_\sigma.\mathbf{c}''_\sigma)$: is proved in a corresponding fashion to $(\mathcal{E}_\sigma.a''_\sigma)$.

References

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

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

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

[4] L. Aceto, A. Ingólfsdóttir, An equational axiomatisation of observation congruence for prefix iteration, in: M. Wirsing, M. Nivat (Eds.), Proceedings of the 5th International Conference on Algebraic Methodology and Software Technology AMAST'96, LNCS, vol. 1101, Springer, 1996.

[5] J.C.M. Baeten, Embedding untimed into timed process algebra: the case for explicit termination, in: L. Aceto, B. Victor (Eds.), Proceedings of the 7th International Workshop on Expressiveness in Concurrency EXPRESS'00, BRICS, 2000.

[6] J.C.M. Baeten, J.A. Bergstra, Real time process algebra, Formal Aspects of Computing 3 (1991) 142–188.

[7] J.C.M. Baeten, J.A. Bergstra, Discrete time process algebra, Formal Aspects of Computing 8 (1996) 188–208.

[8] J.C.M. Baeten, J.A. Bergstra, J.W. Klop, Syntax and defining equations for an interrupt mechanism in process algebra, Fundamenta Informaticae XI (2) (1986) 127–168.

[9] J.C.M. Baeten, C.A. Middelburg, Process algebra with timing: real time and discrete time, in: J.A. Bergstra, A. Ponse, S.A. Smolka (Eds.), Handbook of Process Algebra, Elsevier Science, 2001, pp. 627–684.

[10] J.C.M. Baeten, C.A. Middelburg, Process Algebra with Timing, Springer, 2003.

[11] J.C.M. Baeten, W.P. Weijland, Process Algebra, Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, 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 (1995) 27–68.

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

[16] R.N. Bol, J.F. Groote, The meaning of negative premises in transition system specifications, Journal of the ACM 43 (5) (1996) 863–914.

[17] T. Bolognesi, F. Lucidi, Timed process algebras with urgent interations and a unique powerful binary operator, in: J.W. de Bakker, C. Huizing, W.P. de Roever, G. Rozenberg (Eds.), REX Workshop, LNCS, vol. 600, 1991, pp. 124–148.

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

[19] R.J. van Glabbeek, The linear time-branching time spectrum II, Proceedings of CONCUR'93, LNCS, vol. 715, Springer, Hildesheim.

[20] R.J. van Glabbeek, W.P. Weijland, Branching time and abstraction in bisimulation semantics, in: G.X. Ritter (Ed.), Information Processing 89, 1989, pp. 613–618, JACM.

[21] J.F. Groote, Specification and verification of real time systems in ACP, in: L. Logrippo, L.R. Probert, H. Ural, (Eds.), Proceedings of 10th International Symposium on Protocol Specification, Testing and Verification, 1990.

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

[23] J.F. Groote, The syntax and semantics of timed μCRL, Technical Report SEN-R9709, CWI, Amsterdam, 1997. Available from <ftp://ftp.cwi.nl/pub/CWIreports/SEN/SEN-R9709>.

[24] J.F. Groote, The syntax and semantics of μCRL, in: A. Ponse, C. Verhoef, S.F.M. van Vlijmen (Eds.), Algebra of Communicating Processes'94, Workshops in Computing Series, Springer, 1995, pp. 26–62.

[25] J.F. Groote, F. Vaandrager, Structured operational semantics and bisimulation as a congruence, Information and Computation 100 (1992) 202–260.

[26] M. Hennessy, Timed process algebras: a tutorial, Technical Report 2/93, Department of Computer Science, University of Sussex, 1993.

[27] M. Hennessy, T. Regan, A process algebra for timed systems, Information and Computation 117 (1995) 221–239.

[28] C.A.R. Hoare, Communicating Sequential Processes, Prentice Hall, 1985.
[29] A. Jeffrey, A linear time process algebra, in: K.G. Larsen, A. Skou (Eds.), Proceedings of CAV'91, LNCS, vol. 575, Springer, 1991.
[30] L. Léonard, G. Leduc, An introduction to ET-LOTOS for the description of time-sensitive systems, Computer Networks and ISDN Systems 29 (3) (1997) 271–292.
[31] L. Léonard, G. Leduc, A formal definition of time in LOTOS, Formal Aspects of Computing 10 (1998) 248–266.
[32] R. Milner, A modal characterisation of observable machine behaviours, in: G. Astesiano, C. Böhm (Eds.), Proceedings of CAAP'81, LNCS, vol. 112, Springer, 1981.
[33] R. Milner, Communication and Concurrency, Prentice Hall, 1989.
[34] F. Moller, C. Tofts, A temporal calculus of communicating systems, in: J.C.M. Baeten, J.W. Klop (Eds.), Proceedings of CONCUR'90, LNCS, vol. 458, Springer, 1990.
[35] F. Moller, C. Tofts, Relating processes with respect to speed, in: J.C.M. Baeten, J.F. Groote (Eds.), Proceedings of CONCUR'91, LNCS, vol. 527, Springer, 1991.
[36] R. De Nicola, M. Hennessy, Testing equivalences for processes, Theoretical Computer Science 34 (1984) 83–133.
[37] X. Nicollin, J.-L. Richier, J. Sifakis, J. Voiron, ATP: an algebra for timed processes, in: Proceedings of the IFIP TC2 Working Conference on Programming Concepts and Methods, 1990.
[38] X. Nicollin, J. Sifakis, An overview and synthesis on timed process algebras, in: K.G. Larsen, A. Skou (Eds.), Proceedings of CAV'91, LNCS, vol. 575, Springer, 1991.
[39] X. Nicollin, J. Sifakis, The algebra of timed processes, ATP: theory and application, Information and Computation 114 (1994) 131–178.
[40] I.C.C. Phillips, Refusal testing, Theoretical Computer Science 50 (1987) 241–284.
[41] G.D. Plotkin, A structural approach to operational semantics, Technical Report DAIMI FN-19, Aarhus University, 1981.
[42] G.M. Reed, A.W. Roscoe, A timed model for communicating sequential processes, Theoretical Computer Science 58 (1988) 249–261.
[43] A.W. Roscoe, The Theory and Practice of Concurrency, Prentice Hall, 1998.
[44] S.A. Schneider, An operational semantics for timed CSP, Information and Computation 116 (1995) 193–213.
[45] S.A. Schneider, Concurrent and Real-time Systems, Wiley, 2000.
[46] R. de Simone, Higher-level synchronising devices in MEIJE-SCCS, Theoretical Computer Science 37 (1985) 245–267.
[47] I. Ulidowski, Equivalences on observable processes, in: Proceedings of the 7th Annual IEEE Symposium on Logic in Computer Science, IEEE Computer Science Press, 1992, pp. 148–159.
[48] I. Ulidowski, Local testing and implementable concurrent processes, Ph.D. thesis, Imperial College, University of London, 1994.
[49] I. Ulidowski, Finite axiom systems for testing preorder and de Simone process languages, in: M. Wirsing, M. Nivat (Eds.), Proceedings of the 5th International Conference on Algebraic Methodology and Software Technology AMAST'96, LNCS, vol. 1101, Springer, 1996.
[50] I. Ulidowski, Finite axiom systems for testing preorder and De Simone process languages, Theoretical Computer Science 239 (1) (2000) 97–139.
[51] I. Ulidowski, Priority rewrite systems for OSOS process languages, in: R. Amadio, D. Lugiez (Eds.), Proceedings of the 14th International Conference on Concurrency Theory CONCUR 2003, LNCS, vol. 2761, Springer, 2003.
[52] I. Ulidowski, I.C.C. Phillips, Formats of Ordered SOS rules with silent actions, in: M. Bidoit, M. Dauchet (Eds.), Proceedings of the 7th International Conference on Theory and Practice of Software Development TAPSOFT'97, LNCS, vol. 1214, Springer, 1997.
[53] I. Ulidowski, I.C.C. Phillips, Ordered SOS rules and process languages for branching and eager bisimulations, Information and Computation 178 (1) (2002) 180–213.
[54] I. Ulidowski, S. Yuen, Extending process languages with time, in: M. Johnson (Ed.), Proceedings of the 6th International Conference on Algebraic Methodology and Software Technology AMAST'97, LNCS, vol. 1349, Springer, 1997.
[55] I. Ulidowski, S. Yuen, Process languages for rooted eager bisimulation, in: D. Miller, C. Palamidessi (Eds.), Proceedings of the 11th International Conference on Concurrency Theory CONCUR 2000, LNCS, vol. 1877, Springer, 2000.

[56] C. Verhoef, A congruence theorem for structured operational semantics with predicates and negative premises, Nordic Journal of Computing 2 (2) (1995) 274–302.
[57] D. Walker, Bisimulation and divergence, Information and Computation 85 (2) (1990) 202–241.
[58] Y. Wang, Real-time behaviour of asynchronous agents, in: J.C.M. Baeten, J.W. Klop (Eds.), Proceedings of CONCUR'90, LNCS, vol. 458, Springer, 1990.
[59] Y. Wang, A calculus of real time systems, Ph.D. thesis, Chalmers University of Technology, 1991.
[60] W.P. Weijland, Synchrony and asynchrony in process algebra, Ph.D. thesis, University of Amsterdam, 1989.