Towards General Process Languages with Time

Irek Ulidowski
Department of Mathematics and Computer Science
University of Leicester
iu3@mcs.le.ac.uk

Shoji Yuen
Information Engineering Department
Nagoya University
yuen@nuie.nagoya-u.ac.jp

Abstract

We propose a uniform framework, based on the Ordered Structural Operational Semantics (OSOS) approach of Ulidowski and Phillips [43], for extending arbitrary process languages with discrete time. The generality of our framework allows the user to select the most suitable timed process language for a task in hand. This is possible because the user can choose any operators, whether they are standard operators or new application-specific operators, provided that they satisfy two conditions. Firstly, the operators must preserve a timed version of eager bisimulation and, secondly, all processes in the considered process language must satisfy the time determinacy property. Both of these conditions are verified easily by examining the structure of the OSOS rules defining the operators. We also propose several constraints on the type of OSOS definitions for the operators so that several properties, which reflect the nature of time passage, for example the maximal progress and time persistence properties, are satisfied.

1 Introduction

Process languages, for example CCS [25], CSP [20] and ACP [9], 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*, 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 [28, 26, 21, 18, 30, 19, 32, 7], and process languages with *real* (dense) time, for example [34, 49, 36, 6, 21, 22, 23].

We are convinced that in order to increase the usefulness of process languages in software technology we need to develop a theoretically sound framework for extending process languages with various user-friendly and application-specific features. Enhanced Timed-LOTOS (ET-LOTOS) [22, 23] 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. This paper proposes a framework for extending traditional process languages, like CCS and CSP, with user-defined and application-specific operators and with the notion of time. Since we intend our results to be general we make the following assumptions concerning the method for defining process languages, the preferred preorder on processes and the nature of time:

- There are many process operators which are often defined by SOS rules with negative premises, for example sequential composition, priority and action refinement operators. Instead of rules with negative premises we will use equally expressive and intuitive *Ordered* SOS rules [43], an extension of the original SOS rules of Plotkin [33], as our method for defining process languages.
- We shall consider a divergence sensitive version of weak bisimulation relation as our preorder on processes. For reasons of compositionality and modularity, we insist that all the considered operators preserve this preorder.

- After [30, 19], 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 which are able to let time pass—there may be active components unable to let time pass but this will not block the passage of time in the whole system. This property is called *maximal time synchrony*. We will also consider several other properties which reflect the nature of time passage in systems. These properties are henceforth called the *timed properties*.

Timed process languages which can be defined within our framework have two important characteristics. Firstly, they preserve a timed version of *rooted eager bisimulation*, a divergence sensitive version of weak bisimulation studied in [24, 1, 47, 39, 42, 43, 45]. Secondly, they satisfy the *time determinacy* property. Informally, a timed process language has the above property if every process, by performing a timed action, evolves to a unique process. We will also consider 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 powerful and flexible specification formalism for timed concurrent systems. It subsumes our preliminary paper [44]. Given a system to specify, the user is free to choose a timed process language, which is most 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, 40, 41] can be used, in conjunction with over techniques, to generate sound and complete axiom systems for the chosen timed process language. This may assist the verification stage of system's development.

The paper is organised as follows. In Section 2 we recall the definitions of labelled transition system and the eager bisimulation relation. We also define the rooted eager bisimulation relation. Then, also in Section 2, we introduce OSOS process languages, in particular 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 time. In Section 4 we discuss several timed properties and establish conditions under which these properties are satisfied. In Section 5 we review several existing timed process languages and show how they can be redefined in our framework. The last section contains conclusions and discusses directions for future research.

2 Preliminaries

2.1 Eager and Rooted Eager Bisimulations

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.

$\mathcal{P}$ 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. τ is the silent action and σ is the timed action which denotes 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 role.

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 $p \not\xrightarrow{\chi}$ otherwise. Expressions $p \xRightarrow{\tau} q$ and $p \xRightarrow{\chi} q$, where $\chi \neq \tau$, denote $p(\xrightarrow{\tau})^* q$ and $p(\xrightarrow{\tau})^* \xrightarrow{\chi} q$ respectively. Note, that for technical reasons that we explain shortly our $\xRightarrow{\chi}$ has a different meaning from that originally given by Milner in, for example, [25] 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 $1 \leq i \leq n$, we say that q is an s-derivative of p if $p \xRightarrow{\chi_1} \cdots \xRightarrow{\chi_n} q$. 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 $p \xrightarrow{\hat{\chi}} p'$ 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.

$$
\begin{array}{llll}
(E.a) & \forall \chi.\ (p \xrightarrow{\chi} p' & \text{implies} & \exists q', q''.\ (q \xRightarrow{\tau} q' \xrightarrow{\hat{\chi}} q'' \ \text{ and } \ p'Rq'')) \\
(E.b) & p\Downarrow & \text{implies} & q\Downarrow \\
(E.c) & p\Downarrow & \text{implies} & \forall \chi.\ (q \xrightarrow{\chi} q' \text{ implies } \exists p', p''.\ (p \xRightarrow{\tau} p' \xrightarrow{\hat{\chi}} p'' \ \text{ and } \ p''Rq'))
\end{array}
$$

$p \sqsubseteq\!\!\!\!\sim q$ 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, $q \sqsubseteq\!\!\!\!\sim p$ but $p \not\sqsubseteq\!\!\!\!\sim q$. 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 [25] ($s = r$ is an instance of the third τ-law), but $r \not\sqsubseteq\!\!\!\!\sim s$ since after $s \xrightarrow{a} c$ there is no r' such that $r \xRightarrow{\tau}\xrightarrow{a} r'$ and $r' \sqsubseteq\!\!\!\!\sim c$.

It is clear that $\sqsubseteq\!\!\!\!\sim$ is a preorder. $\sqsubseteq\!\!\!\!\sim$ is the version of weak bisimulation studied in [24, 1, 47, 39, 42, 43], where testing, modal logic and axiomatic characterisations were proposed and congruence results with respect to the ISOS, eb and ebo formats were proved. $\sqsubseteq\!\!\!\!\sim$ coincides with *delay* bisimulation [50, 16] for processes with no divergence. We have chosen this finer version of weak bisimulation in preference to the standard weak bisimulation [25] because formats for eager bisimulation [38, 39, 42, 43] are simpler than those for the standard weak bisimulation [11]. Moreover, since the formats for eager bisimulation allow rules with negative premises [38, 39] or rules with orderings [42, 43], which have the effect of negative premises, they are more general than the formats for weak bisimulation [11]. 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 [43] 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 $1 \leq i \leq n$, 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 $+$ and the ACP left-merge $\lfloor\!\lfloor$ operators. We have $a \sqsubseteq\!\!\!\!\sim \tau.a$ but neither $a+b \sqsubseteq\!\!\!\!\sim \tau.a+b$ nor $a \lfloor\!\lfloor\, b \sqsubseteq\!\!\!\!\sim \tau.a \lfloor\!\lfloor\, b$. The standard solution to this problem is to use the *rooted* version of the preferred relation instead the relation itself [25, 9]. 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.

$$(R.a) \quad \forall \alpha.\ (p \xrightarrow{\alpha} p' \quad \text{implies} \quad \exists q', q''.\ (q \stackrel{\tau}{\Rightarrow} q' \xrightarrow{\alpha} q'' \ \text{ and } \ p' \lesssim q''))$$
$$(R.c) \qquad\qquad p\Downarrow \quad \text{implies} \quad \forall \alpha.\ (q \xrightarrow{\alpha} q' \text{ implies } \exists p', p''.\ (p \stackrel{\tau}{\Rightarrow} p' \xrightarrow{\alpha} p'' \ \text{ and } \ p'' \lesssim q'))$$

$p \lesssim_r q$ if there exists a rooted eager bisimulation R such that pRq.

It is easy to show that $\lesssim_r$ 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 $p \lesssim_r q$ then $p\Downarrow$ implies $q\Downarrow$.

In Section 3 and afterwards we shall use actions 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, $\lesssim_r$ 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, which do not preserve $\lesssim_r$. Thus, we will use a slight alteration of $\lesssim_r$, called *timed rooted eager bisimulation*, as our preferred preorder on timed processes.

2.2 Ordered SOS Process Languages

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

The set of *closed terms*, written as $\mathrm{T}(\Sigma)$, is ranged over by $p, q, u, v, \ldots$. In the setting of process languages these terms will often be called process terms. A Σ context with n holes $C[X_1, \ldots, X_n]$ is a member of $\mathbb{T}(\Sigma, \{X_1, \ldots, X_n\})$. If $t_1, \ldots, t_n$ are Σ terms, then $C[t_1, \ldots, t_n]$ is the term obtained by substituting t_i for X_i for $1 \leq i \leq n$.

We will use bold italic font to abbreviate the notation for sequences. For example, a sequence of process terms $p_1, \ldots, p_n$, for any $n \in \mathbb{N}$, will often be written as $\boldsymbol{p}$ when the length is understood from the context. Given any binary relation R on closed terms and $\boldsymbol{p}$ and $\boldsymbol{q}$ of length n, we will write $\boldsymbol{pRq}$ to mean $p_i R q_i$ for all $1 \leq i \leq n$. Moreover, instead of $f(X_1, \ldots, X_n)$ we will often write $f(\boldsymbol{X})$ when the arity of f is understood.

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

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

The notion of transition is extended to expressions $t \xrightarrow{\chi} t'$.

Definition 5 A (transition) rule, or an SOS rule, is an expression of the form

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

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. Variables in $\boldsymbol{X}$ are the *arguments* of f. Moreover, $I \subseteq \{1, \ldots, n\}$ and all J_i are finite subsets of $\mathbb{N}$, and $C[\boldsymbol{X}, \boldsymbol{Y}]$ is a context.

Let r be the above rule. Then, f is the *operator* of r and $rules(f)$ is the set of all rules with the operator f. The set of transitions above the horizontal bar in r is called the *premises*, written as $pre(r)$. The transition below the bar in r is the *conclusion*, written as $con(r)$. χ in the conclusion of r is the *action* of r, written as $act(r)$, and $C[\boldsymbol{X},\boldsymbol{Y}]$ is the *target* of r. 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 $\sigma = act(r)$ or $\sigma \in actions(r)$, then r is a *timed* rule. If $\sigma = act(r)$ and $actions(r) = \{\sigma\}$, then r is a σ-*rule*. A rule is a *visible action* rule, or simply action rule, if $\tau \notin actions(r)$.

The rules in Definition 5 belongs to the GSOS format [12, 13]. GSOS rules may also have *negative* premises, i.e. expressions of the form $X_k \stackrel{\chi_{kl}}{\nrightarrow}$ for appropriate k and l: see Example 9 below.

Next, we recall the notion of ordering on rules as in [43]. Let $<_f$ be a binary relation on $rules(f)$. 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 a signature Σ, the relation $<_\Sigma$, or simply $<$ if Σ is understood from the context, is defined as $\bigcup_{f\in\Sigma} <_f$. $<_\Sigma$ is called the *ordering* on the rules for the operators in Σ. We will write $higher(r)$ for $\{r' \mid r < r'\}$. We make no assumptions about $<$. A discussion about possible properties of $<$ and how they effect the expressive power of SOS rules with $<$ in defining processes operationally can be found in [43].

Definition 6 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 such that all actions mentioned in the rules belong to A, and $<$ is the ordering on the rules for the operators in Σ.

Given an OSOS process language $G = (\Sigma, A, R, <)$, we recall how to associate a unique transition relation $\to$ with G [43]. Let $d : \mathrm{T}(\Sigma) \to \mathbb{N}$ 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 $d(f(p_1,\ldots,p_n)) = 1 + \max\{d(p_i) \mid 1 \le i \le n\}$ otherwise.

Definition 7 Given an OSOS process language $(\Sigma, A, R, <)$, we associate with it a transition relation, $\to \subseteq \mathrm{T}(\Sigma) \times A \times \mathrm{T}(\Sigma)$, which is defined as $\to = \bigcup_{l<\omega} \to^l$, where transition relations $\to^l \subseteq \mathrm{T}(\Sigma) \times A \times \mathrm{T}(\Sigma)$ are defined as follows:

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

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

Definition 8 Let $r \in rules(f)$ and $pre(r) = \{X_i \stackrel{\alpha_{ij}}{\to} Y_{ij} \mid i \in I, j \in J_i\}$. Rule r *applies* to $f(\boldsymbol{u})$ if and only if $u_i \stackrel{\alpha_{ij}}{\to}$ 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, we easily obtain $(\mathrm{T}(\Sigma), A, \to)$ as the LTS for G. Thus, eager bisimulation, its rooted and timed rooted versions are defined over this LTS as in Definitions 2, 4 and 17.

It is argued in [43] that OSOS process languages and GSOS process languages [14] have the same expressive power. Hence, instead of GSOS rules, which may have negative premises, i.e. expressions of the form $X \stackrel{\chi}{\nrightarrow}$ in the premises, we will use equally expressive ordered rules for defining process operators.

Example 9 Operational definitions in terms of ordered rules have the same effect as those in terms of (GSOS) rules with negative premises. Consider the sequential composition operator ; discussed in [14]. It is defined by two GSOS rule schemas, for any α and β:

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

Alternatively ; can be defined by simple OSOS rules

$$\frac{X \xrightarrow{\alpha} X'}{X;Y \xrightarrow{a} X';Y}\; r'_{\alpha*} \qquad\qquad \frac{Y \xrightarrow{\beta} Y'}{X;Y \xrightarrow{\beta} X;Y'}\; r'_{*\beta}$$

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

Finally, we introduce a general class of OSOS process languages whose operators preserve $\underset{\sim}{\sqsubset}_r$. Since an arbitrary OSOS process language may contain operators which do not preserve $\underset{\sim}{\sqsubset}_r$ we define several conditions on the structure of rules and on the orderings which guarantee that $\underset{\sim}{\sqsubset}_r$ is preserved. We will need the following notions of *active arguments*, *silent rules* and *copies* in transition rules.

The ith argument X_i is *active* in a rule r, written as $i \in \mathit{active}(r)$, if it appears in the premises of r [10]. Overloading the notation we write $\mathit{active}(f)$ instead of $\{i \mid i \in \mathit{active}(r) \text{ and } r \in \mathit{rules}(f)\}$. Consequently, the ith argument of $f(\boldsymbol{X})$ is active if it is active in some rule for f.

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* and define them as all rules r such that $\mathit{act}(r) = \tau$ and $\mathit{actions}(r) = \{\tau\}$. The most widely used silent rules are the τ-rules, also called *patience* rules, [10]. 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, is as follows:

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

Given a rule r for f, the set of τ-rules for r consists of all τ_i such that $i \in \mathit{active}(r)$. Thus, the set of τ-rules for f consists of all τ_i such that $i \in \mathit{active}(f)$. 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, is as follows:

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

We shall write $\mathit{tau}^f(i)$, or simply $\mathit{tau}(i)$ if f clear from the context, to denote either τ_i^f or $\tau^{f,i}$.

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:

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

- τ-rules for all active arguments:

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

- no τ actions in the premises of rules except for τ-rules:

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

- no rules above their associated τ-rules:

$$\textbf{if } \; r' < r \;\; \textbf{and } \; i \in \mathit{active}(r) \;\; \textbf{then } \; r' < \tau_i \tag{3}$$

- rules and their τ-rules below a rule:

$$\textbf{if } \; \tau_i < r \;\; \textbf{and } \; i \in \mathit{active}(r') \cup \mathit{active}(\mathit{higher}(r')) \;\; \textbf{then } \; r' < r \tag{4}$$

- τ-rules are always enabled:

$$\textbf{not } (\tau_i < \tau_i) \tag{5}$$

- rules with implicit copies are allowed provided they are below their τ-rules:

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

Figure 1: Conditions for τ-preserving operators

Examples of choice rules are the rules for + of CCS, the rule for the left-merge operator, and the first two rules for the delay operator in Example 16.

Multiple occurrences of process variables in rules are called *copies*. They can be divided into *explicit* and *implicit* copies [39]. Given a rule r as in Definition 5, explicit copies are the multiple occurrences of variables Y_{ij} and X_i, for $i \notin I$, in the target t. The implicit copies are the multiple occurrences of X_i in the premises of r and the occurrences, not necessarily multiple, of variables X_i in t when $i \in I$. The set of all implicit copies of process variables in a rule r for f is denoted by $\mathit{implicit\text{-}copies}(r, f)$.

We are ready to define a general class of OSOS process languages whose operators preserve $\sqsubseteq_r$. Consider conditions in Figures 1 and 2, where f is an arbitrary operator of an OSOS process language, $<$ is the ordering on the rules for the language and thus the ordering on the rules for f, rules r and r' are typical rules in $\mathit{rules}(f)$, and τ_i and τ^i are the τ-rule and the silent choice rule for the ith argument of f, respectively.

Definition 10 An operator f is τ*-preserving* if the set of its rules and the ordering on the rules satisfy (1)–(6). An operator f is τ*-sensitive* if the set of its rules and the ordering on the rules satisfy (6)–(12). An OSOS process language is *rooted eager bisimulation ordered* (`rebo`) if its signature can be partitioned into τ-preserving and τ-sensitive operators and the targets of all rules, except for the τ-rules of τ-sensitive operators, contain only τ-preserving operators. Operators of `rebo` process languages are called `rebo` operators.

Conditions in Fig. 1 were introduced in [43] in order to define a subclass of OSOS process languages that preserve eager bisimulation relation. Examples were also given there to demonstrate that the conditions could not be relaxed without compromising the congruence result. Need more!!!

The τ-rules for any operator f that are guaranteed by condition (1) shall be henceforth called the τ-rules *associated with* f. Similarly, given any rule r for f, all τ-rules τ_i for f such that $i \in \mathit{active}(r)$ shall be called the τ-rules *associated with* r.

- either a silent choice rule or a τ-rule for each active argument:

$$\textbf{if } \ i \in active(f) \ \textbf{ then } \ (\textbf{either } \tau_i \in rules(f) \ \textbf{ or } \ \tau^i \in rules(f)) \tag{7}$$

- only choice rules have no associated τ-rules:

$$\textbf{if } \ i \in active(f) \ \textbf{ and } \ \tau_i \notin rules(f) \ \textbf{ then } \ r \text{ is a choice rule for } f \text{ and } i \tag{8}$$

- no τ actions in the premises of rules except for silent rules:

$$\textbf{if } \ \tau \in action(r,i) \ \textbf{ then } \ (r = \tau_i \ \textbf{ or } \ r = \tau^i) \tag{9}$$

- no rules above their silent rules:

$$\textbf{if } \ r' < r \ \textbf{ and } \ i \in active(r) \ \textbf{ then } \ (r' < \tau_i \ \textbf{ or } \ r' < \tau^i) \tag{10}$$

- rules and their silent rules below a rule:

$$\textbf{if } \ (\tau_i < r \ \textbf{ or } \ \tau^i < r) \ \textbf{ and } \ i \in active(r') \cup active(higher(r')) \ \textbf{ then } \ r' < r \tag{11}$$

- silent rules are always enabled:

$$\textbf{not } (\tau_i < \tau_i) \ \textbf{ and not } (\tau^i < \tau^i) \tag{12}$$

Figure 2: Conditions for τ-sensitive operators

rebo process languages are descendants of process languages for eager bisimulation preorder introduced in [42, 43]. 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 [11]. It has also been employed in [15, 43]. Most of the popular process operators are rebo: they are either τ-preserving or τ-sensitive, where the second group contains the CCS +, the ACP left-merge, the Kleene star [4, 3] and Milner's interrupt operators [25]. One exception of a popular operator which is not rebo is the version of the Kleene star operator in [15].

Finally, we present the main result of this section. The importance of this result is that it permits compositionality of specifications and modularity of verification of systems described in rebo process languages. The proof is given in Appendix A. It follows the same lines as the proof of the congruence result for ebo process languages and eager bisimulation in [43].

Theorem 11 All rebo process languages preserve rooted eager bisimulation.

3 Timed Process Languages

In this section we propose a uniform method for extending an arbitrary rebo process language with the notion of discrete 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 two steps as follows:

1. We augment the definitions of the operators from the given (un-timed) rebo process language by adding appropriate timed rules so that the passage of time is preserved. Operators extended in such way will be called *time preserving* operators.

2. We add new process operators which introduce, alter or stop the passage of time in processes. Such operators will be called *time altering* operators.

In the rest of this section, we define the forms of rules for time preserving and time altering operators, define the *timed rooted eager bisimulation* relation and timed process languages, and, finally, we prove that all timed process languages preserve timed rooted eager bisimulation and that their processes are time deterministic.

3.1 Time Preserving Operators

We propose a method for extending the definitions of rebo operators with the notion of discrete time so that the resulting operators preserve the passage of time. Following [34, 49, 29], one may expect that, given a rebo operator f, it is sufficient to extend the set of rules for f with the following σ-rule

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

where $J = \mathit{active}(f)$ and $Y_i = X_i$ for $i \notin \mathit{active}(f)$.

Notation. A σ-rule of the above form will be denoted thereafter by σ^f_J, where J indicates which arguments appear in the premises, or simply by σ_J when the operator is obvious from the context. When $J = \{j\}$, for some $j \in \mathbb{N}$, then we simply write σ_j for σ_J.

This extension method works for the + and the parallel composition operators of CCS, and for many other process operators: it guarantees that the maximal time synchrony property holds. But, the method is not suitable for some operators which are defined by ordered rules. For example, for the sequential composition operator, which we have defined in Example 9, we obtain 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 maximal 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 (4) and (3) 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 disallow $\sigma_{\{1,2\}}$ and, instead of it, to add two σ-rules, one for each of ; arguments, and order them accordingly.

We will need the following auxiliary notions. By conditions (1) and (7), there must be either a τ-rule, τ_i, or a silent choice rule, τ^i, for each active argument X_i of f. Recall that $\mathit{tau}(i)$ denotes either τ_i or τ^i. We say that two active arguments X_i and X_j of f have the *same priority* if and only if $\mathit{tau}(i) \not< \mathit{tau}(j)$ and $\mathit{tau}(j) \not< \mathit{tau}(i)$. If $\mathit{active}(f) \neq \emptyset$, then let $\mathit{Levels}(f)$ be the set of all non-empty subsets K of $\mathit{active}(f)$ such that, for all K, all members of K have the same priority. 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 $K \cup L \in \mathit{Levels}(f)$. When $\mathit{active}(f) = \emptyset$, then we set $\mathit{Levels}(f)$ to $\{\emptyset\}$. Members of $\mathit{Levels}(f)$ are called the *priority levels* for f.

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 $\prec$.

There are several useful results concerning priority levels. If $K \subset L$, then $K \prec_f L$; also $K \not\prec_f K$ for all K. If K and L are priority levels, then either $L \subseteq K$, or $K \subseteq L$, or $K \cup L = \emptyset$. Next, we define a new notion. Let $K \downarrow L = \{k \mid tau(k) < tau(l) \text{ for no } l \in L\}$, in other words, $K \downarrow L$ the set of all k in K which do not have lower priority than any members of L. Note, that when $K \subseteq L$, we have $K \downarrow L = K$ and $L \downarrow K = L$ as all members of L, and thus K, have the same priority.

We have the following straightforward result.

Lemma 12 If K and L are priority levels, then $M = K \downarrow L \,\cup\, L \downarrow K$ is also a priority level.

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 $K \subseteq L$, we have $\mathsf{lupl}(K,L) = L$. So, for $K \neq L$, $K \prec \mathsf{lupl}(K,L)$ but not $L \prec \mathsf{lupl}(K,L)$ since $L = \mathsf{lupl}(L,K)$. Moreover, if $K \cup L = \emptyset$, then $K \prec \mathsf{lupl}(K,L)$ and $L \prec \mathsf{lupl}(L,K)$. Assume for contradiction that $k \in K$, $tau(k) < tau(l)$ for some $l \in L$, and $l \notin \mathsf{lupl}(K,L)$. The last is only valid when $tau(l) < tau(k')$ for some $k' \in K$. By condition (4) or (11), $tau(l) < tau(k')$ and $l \in active(higher(tau(k)))$ implies $tau(k) < tau(k')$, which contradicts the fact that k and k' have the same priority.

Definition 13 Let $L = (\Sigma, A, R, <)$ be rebo process language with $\sigma \notin A$. The *time preserving extension* of L, $(\Sigma', A', R', <')$, is as follows:

1. $\Sigma' = \Sigma$.
2. $A' = A \cup \{\sigma\}$.
3. $R' = R \cup \bigcup_{f \in \Sigma} R^f_\sigma$, where $R^f_\sigma = \{\sigma^f_J \mid J \in Levels(f)\}$.
4. $<' = \bigcup_{f \in \Sigma} <'_f$, where each $<'_f$ is the least ordering that contains $<_f \cup <^{\sigma,\sigma}_f \cup <^{\tau,\sigma}_f$ and satisfies conditions (3) and (4) if f is τ-preserving, and conditions (10) and (11) otherwise, where $<^{\sigma,\sigma}_f$ and $<^{\tau,\sigma}_f$ are

   (4.a) $<^{\sigma,\sigma}_f = \{(r,r') \mid r,r' \in R^f_\sigma \wedge active(r) \prec_f active(r')\}$,

   (4.b) $<^{\tau,\sigma}_f = \{(tau^f(k), r) \mid r \in R^f_\sigma \wedge \forall l \in active(r).\ tau^f(k) < tau^f(l)\}$.

The operators of process languages extended in the above way are called time preserving operators. Process languages consisting of time preserving operators are called time preserving process languages.

Note, that since timed rules of time preserving operators are σ-rules the orderings in (4.a) and (4.b) above can be defined more specifically as $<^{\sigma,\sigma}_f = \{(\sigma^f_K, \sigma^f_L) \mid K,L \in Levels(f) \wedge K \prec_f L\}$ and as $<^{\tau,\sigma}_f = \{(tau^f(k), \sigma^f_L) \mid \forall l \in active(r).\ tau^f(k) < tau^f(l)\}$. However, due to the existance of corresponding orderings in Definition 15 below, we keep the more general form of definitions.

Example 14 Time preserving versions of popular process operators are obtained simply by adding the required σ-rules for each of their priority levels. For example, for the + and parallel composition operators of CCS we add the following σ-rules for the priority levels $\{1,2\}$, $\{1\}$ and $\{2\}$.

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

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

Since $\{1\} \prec \{1,2\}$ and $\{2\} \prec \{1,2\}$ we have $\sigma_1^{\|} < \sigma_{\{1,2\}}^{\|}$ and $\sigma_2^{\|} < \sigma_{\{1,2\}}^{\|}$ by (4.a). For the same reason we also have $\sigma_1^{+} < \sigma_{\{1,2\}}^{+}$ and $\sigma_2^{+} < \sigma_{\{1,2\}}^{+}$ by (4.a). Moreover, we have $\sigma_1^{\|}, \sigma_2^{\|} < \tau_1, \tau_2$ by condition (3), and $\sigma_1^{+}, \sigma_2^{+} < \tau^1, \tau^2$ by (10). Notice that the rules and the orderings above guarantee that when only one of the component processes in $p+q$ or $p \parallel q$ blocks time and it stable, the composite processes can still pass time provided the other component is also stable and does not block time. Thus, **stable** maximal time synchrony holds.

The sequential composition operator ; has two priority levels, namely $\{1\}$ and $\{2\}$, so to make it time preserving we add two σ-rules:

$$\frac{X \stackrel{\sigma}{\to} X'}{X;Y \stackrel{\sigma}{\to} X';Y}\ \sigma_1 \qquad\qquad \frac{Y \stackrel{\sigma}{\to} Y'}{X;Y \stackrel{\sigma}{\to} X;Y'}\ \sigma_2$$

Since $\{2\} \prec \{1\}$ the new ordering on the rules for the time preserving version of ; contains $\sigma_2 < \sigma_1$ by (4.a). As ; is τ-preserving, condition (4) requires $\sigma_2 < r_{a*}$ since $\tau_2 < r_{a*}$. Also, (3) requires $\sigma_2 < \tau_1$ since $\sigma_2 < r_{a*}$. Moreover, we also need $\tau_2 < \sigma_1$ by (4.b), and $r_{*a} < \sigma_1$ by condition (4).

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

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, it appears in the target of the conclusion of its σ-rules (see Definition 10).

There are only few timed versions of traditional process operators in the concurrency literature which are not time preserving. The examples are the ACP + in [17] and $\oplus$ in [26]; and the resulting LTSes for these time process languages are *not* time deterministic.

3.2 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 the rules. We motivate and derive these restrictions below.

The simplest operator that introduces the passage of time is a *delay* operator $\sigma.X$ [49, 30, 19, 7] defined as follows:

$$\frac{}{\sigma.X \stackrel{\sigma}{\to} X}$$

Clearly, $\sigma.$ is rebo. There are many other operators that introduce or alter the passage of time [34, 49, 36, 6, 18, 30, 19, 7].

In general, when defining time altering operators we will not use rules which change ordinary actions into timed actions or vice versa, i.e. rules with the action σ and visible or silent actions in the premises (or vice versa). This is because it is possible to define new operators with such rules so that the operators do not preserve $\precapprox_{tr}$ (Definition 17 below) and time determinacy: Let f be an operator which renames action σ of its only operand to a and leaves other actions unchanged. Let $p = \sigma.b + \sigma.c$ and $q = \sigma.(b+c)$, where + is the time preserving version of the CCS +. We clearly have $p \precapprox_r q$—this embodies the time determinacy property in the timed extensions of CCS-like languages. But, $f(p) \precapprox_r f(q)$ does not hold because $ab + ac \not\precapprox_r a(b+c)$. Next, consider an operator g which renames action a of its only argument to σ and leaves other actions intact. Operator g is unsafe because, although it preserves $\precapprox_r$, it does not preserve time determinacy. For, $g(ab+ac)$ can evolve by σ to syntactically different processes, namely b and c. Thus, we demand 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.

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

Consider $h(p,q)$ with $p \overset{\sigma}{\to} p'$, $q \overset{\sigma}{\to} q'$ and $p' \neq q'$. Clearly, if neither $r_1 < r_2$ nor $r_2 < r_1$, then $h(p,q)$ is not time deterministic. Our solution for operators like h involves two alternatives: either there is a timed rule $r_{\{1,2\}}$ for f in addition to the rules above with $r_1, r_2 < r_{\{1,2\}}$, or $t_1 = t_2$ up to a change of process variables. A general formulation of the first alternative will require the usage of the predicate lupl.

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, either there is only one timed rule enabled at $f(\boldsymbol{p})$ or there are several enabled timed rules but with identical conclusions up to a change of process variables. This is precisely the motivation for condition (13) in Definition 15: 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 of fixed form. Secondly, the ordering on timed rules for time altering operators are less restrictive than those for time preserving operators.

Definition 15 Let $L = (\Sigma, A, R, <)$ be a rebo process language with $\sigma \notin A$. The *time altering extension* of L, $(\Sigma', A', R', <')$, is defined as follows, and it satisfies an additional condition (13) given below.

1. $\Sigma' = \Sigma$.
2. $A' = A \cup \{\sigma\}$.
3. $R' = R \cup \bigcup_{f \in \Sigma} R^f_\sigma$, where $R^f_\sigma \subseteq \{r \mid r \text{ is a timed rule } \wedge (active(r) \in Levels(f) \vee active(r) = \emptyset)\}$.
4. $<' = \bigcup_{f \in \Sigma} <'_f$, where each ordering $<'_f$ contains $<_f \cup <^{\sigma,\sigma}_f \cup <^{\tau,\sigma}_f$ and satisfies conditions (3), (4) and (6) if f is τ-preserving, and conditions (10), (11) and (6) otherwise. Orderings $<^{\sigma,\sigma}_f$ and $<^{\tau,\sigma}_f$ are defined as follows:

   (4.a) $<^{\sigma,\sigma}_f = \{(r,r') \mid r,r' \in R^f_\sigma \wedge active(r) \prec active(r')\}$,

   (4.b) $<^{\tau,\sigma}_f = \{(tau^f(k), r) \mid r \in R^f_\sigma \wedge \forall l \in active(r).\ tau^f(k) < tau^f(l)\}$.

For all f in Σ, and all r and r' in R^f_σ

$$\textbf{if } r \not<' r' \textbf{ and } r' \not<' r \quad \textbf{then} \quad \exists r'' \in R^f_\sigma.\ (active(r'') = \mathsf{lupl}(active(r),\ active(r')) \textbf{ and } r, r' < r'') \\ \textbf{or } \forall \rho.\ \rho(con(r)) = \rho(con(r'))) \qquad (13)$$

The operators of process languages extended in the above way are called time altering operators. Process languages consisting of time altering operators are called time altering process languages.

Condition (13) has been motivated by the example of operator h above. We need (13) because there are timed operators which do not satisfy it and thus fail time determinacy. The examples are the timed version of the ACP + [17] and the operator $\oplus$ in [26]. The σ-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 $t_1 = X'$ and $t_2 = Y'$. Although the sets of active arguments in the two σ-rules for $+$ are disjoint there is no third σ-rule with the active arguments X and Y, and the conclusions of the two σ-rules for $+$ are not identical under all substitutions as (13) requires.

Example 16 All timed operators discussed in [34, 49, 36, 6, 18, 30, 19, 7] are essentially time altering operators provided that their definitions are slightly changed by including the required τ-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 [19]. We use ordered rules instead of rules with negative premises as used originally. The rules are as follows:

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

The ordering satisfies $\sigma_\emptyset < r_\tau$. We easily check that $\lfloor\ \rfloor(\ )$ is a time altering and τ-sensitive operator. It introduces the initial passage of one time unit only when its first argument is stable, i.e. cannot perform τ actions. Notice that $\lfloor\ \rfloor(\ )$ does not preserve eager bisimulation $\underset{\sim}{\sqsubset}$. But, since $\lfloor\ \rfloor(\ )$ is a rebo operator it preserves $\underset{\sim}{\sqsubset}_r$, and hence $\underset{\sim}{\sqsubset}_{tr}$.

3.3 Timed Process Languages

In this subsection we define a general class of timed process languages whose members preserve timed rooted eager bisimulation and satisfy time determinacy. Then, we identify several subclasses of timed process languages which additionally satisfy some other timed properties (Fig. 3).

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

Definition 17 A binary relation on processes R is a timed rooted eager bisimulation if, for all $(p,q) \in R$, the following properties hold.

$$\begin{array}{llll}
(R_\sigma.a1) & \forall\alpha.\ (p \xrightarrow{\alpha} p' & \text{implies} & \exists q', q''.\ (q \overset{\tau}{\Rightarrow} q' \xrightarrow{\alpha} q'' \text{ and } p' \underset{\sim}{\sqsubset} q'')) \\
(R_\sigma.a2) & p \xrightarrow{\sigma} p' & \text{implies} & \exists q'.\ (q \xrightarrow{\sigma} q' \text{ and } p'Rq') \\
(R_\sigma.c1) & p\Downarrow & \text{implies} & \forall\alpha.\ (q \xrightarrow{\alpha} q' \text{ implies } \exists p', p''.\ (p \overset{\tau}{\Rightarrow} p' \xrightarrow{\alpha} p'' \text{ and } p'' \underset{\sim}{\sqsubset} q')) \\
(R_\sigma.c2) & p\Downarrow & \text{implies} & (q \xrightarrow{\sigma} q' \text{ implies } \exists p'.\ (p \xrightarrow{\sigma} p' \text{ and } p'Rq'))
\end{array}$$

$p \underset{\sim}{\sqsubset}_{tr} q$ if there exists a timed rooted eager bisimulation R such that pRq.

Definition 18 A process language is called timed rebo, or simply *timed process language*, if it can be partitioned into process languages L_1 and L_2 with no common operators such that, L_1 is a time preserving process language, L_2 is a time altering process language, and the targets of all σ-rules in L_2 include only τ-preserving operators.

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 τ-sensitive as, for example, time preserving +. Thus, in general, timed rebo operators do not preserve rooted eager bisimulation. Consider time preserving +. We have $\sigma.a \underset{\sim}{\sqsubset}_r \sigma.\tau.a$ but $p = \sigma.a + \sigma.b \not\underset{\sim}{\sqsubset}_r \sigma.\tau.a + \sigma.b = q$ since $p \xrightarrow{\sigma} a+b$ and $q \xrightarrow{\sigma} \tau.a+b$, and clearly $a+b \not\underset{\sim}{\sqsubset} \tau.a+b$. However, because of the form of σ-rules for time preserving operators (Definition 13) and a suitably defined timed rooted eager bisimulation (Definition 17) we have the following results.

Theorem 19 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 $p' = p''$.

If p is a constant, say f, then its SOS rules, including timed rules, have no premises and thus no active arguments. When f is time preserving, it has only one timed rule: $\sigma_{\emptyset}^{f} : f \xrightarrow{\sigma} f$, and time determinacy holds. When f is time altering, it may have several timed rules, possibly ordered, according to Definition 15. Let $f \xrightarrow{\sigma} f'$ and $f \xrightarrow{\sigma} f''$ be derivable by timed rules r and r', respectively. Hence, r and r' are enabled at f, so $r \not< r'$ and $r' \not< r$. Now, we make use of condition (13). There cannot be a timed rule r'' such that $r, r' < r''$, because if there was, it would have been enabled at f, thus contradicting the assumption that r and r' are enabled. So, we deduce $con(r) = con(r')$, and $f' = f''$ as required.

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'. There are two cases to consider.

1. f is time preserving. Because of the form of σ-rules it is sufficient to prove that there cannot be two σ-rules enabled at $f(\boldsymbol{u})$, with $K, L \neq \emptyset$ and $K \neq L$. Suppose for contradiction that different σ-rules σ_K and σ_L are enabled at $f(\boldsymbol{u})$. If $K \prec L$, then $\sigma_K < \sigma_L$ and σ_K is not enabled at $f(\boldsymbol{u})$: contradiction. Without loss of generality assume that neither $K \prec L$ nor $L \prec K$ holds. This implies $K \not\subset L$ and $L \not\subset K$. We have two cases:

   Firstly, there is no $k \in K$ such that $tau(k) < tau(l)$ for some $l \in L$, and no $l \in L$ such that $tau(l) < tau(k)$ for some $k \in K$. This means that the elements of K and L have the same priority. Thus, $K \cup L$ a priority level for f, and $\sigma_L, \sigma_K < \sigma_{L \cup K}$. Since $\sigma_{L \cup K}$ is applicable to $f(\boldsymbol{u})$, either $\sigma_{L \cup K}$ itself or a timed rule higher than $\sigma_{L \cup K}$ is enabled at $f(\boldsymbol{u})$: contradiction.

   Secondly, there is $k \in K$ such that $tau(k) < tau(l)$ for some $l \in L$. Let K' be the set of all $k \in K$ such that $tau(k) < tau(l)$ for the above l. The set $K \setminus K' \cup \{l\}$ is a priority level for f since there is no $k' \in K$ such that $tau(l) < tau(k')$: assume for contradiction that such k' exists. Then, by condition (4), $tau(k) < tau(k')$, for all $k \in K'$, which contradicts the assumption that such k and k' belong to the same priority level. Clearly, $K \prec K \setminus K' \cup \{l\}$, so $\sigma_{K \setminus K' \cup \{l\}}$ or a higher timed rule is enabled at $f(\boldsymbol{u})$: contradiction.

   Hence, there is at most one σ-rule enabled at $f(\boldsymbol{u})$. Let σ_L be this rule. By the induction hypothesis the subterms u_i are time deterministic. Hence, $f(\boldsymbol{u})$ has a unique σ-derivative.

2. f is time altering. Suppose $f(\boldsymbol{u}) \xrightarrow{\sigma} p'$ is derived by a timed rule r and $f(\boldsymbol{u}) \xrightarrow{\sigma} p''$ by a time rule r'. Since both r and r' are enabled at $f(\boldsymbol{u})$, we deduce $r \not< r'$ and $r' \not< r$. Hence, by condition (13), either $p' = p''$, which completes the proof, or there is a timed rule r'' for f such that $active(r'') = active(r) \cup active(r')$ and $r, r' < r''$. With our assuptions as above, the second option is not possible: Assume the opposite for contradiction. Since r and r'' are applicable to $f(\boldsymbol{u})$ we deduce that r'' is applicable to $f(\boldsymbol{u})$. Hence, either r'' itself or a timed rule higher than r'' is enabled at $f(\boldsymbol{u})$. This contradicts the assumption that r and r' are enabled at $f(\boldsymbol{u})$.

$\square$

Next, we have the congruence theorem for the timed process languages. The proof is similar to that of Theorem 11, and is given in Appendix B.

Theorem 20 All timed process languages preserve timed rooted eager bisimulation.

4 Properties of Timed Process Languages

There are several other timed properties, which reflect the nature of time passage, which commonly appear in the concurrency literature, for example [49, 29, 36, 21, 18]. We list the properties that we shall consider in Fig. 3. For brevity, we leave out the outermost universal quantifies binding p, where appropriate. There are other timed properties which are not considered in this paper. Firstly, there is the

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** $p \xRightarrow{\sigma}$
maximal progress	**if** $p \xrightarrow{\tau}$ **then** $p \stackrel{\sigma}{\nrightarrow}$
patience	**if** $p \stackrel{\tau}{\nrightarrow}$ **then** $p \xrightarrow{\sigma}$
time persistence	**if** $p \xrightarrow{a}$ **and** $p \xrightarrow{\sigma} p'$ **then** $p' \xrightarrow{a}$, for any a
urgency	$q \stackrel{\tau}{\nrightarrow}$ **and** $q \stackrel{\sigma}{\nrightarrow}$, for some q

Figure 3: Timed properties

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 in terms of time units, the additivity is trivially satisfied. Secondly, these are *finite* and *bounded variability* properties [34, 29].

There are dependencies between some of the properties in Fig. 3.

Lemma 21 The negation of the urgency property is equivalent to the patience property, and the patience property implies the weak timelock freeness property.

4.1 Maximal Progress

A process satisfies maximal progress if whenever it is able to evolve silently by performing a τ, then it must not be able to pass time. Equivalently, whenever it is able to pass time, then it must be stable (if $p \xrightarrow{\sigma}$, then $p \stackrel{\tau}{\nrightarrow}$).

Theorem 22 Let L be any timed process language and T be its LTS. If all operators f in L satisfy (14) defined below, then T satisfies the maximal progress property.

For all rules r and timed rules r_σ for f, and all $J \in Levels(f)$

$$\textbf{if } act(r) = \tau \textbf{ and } active(r) \cup active(r_\sigma) \subseteq J \textbf{ then } r_\sigma < r \tag{14}$$

Proof. Let f be an operator of L and let $J \in Levels(f)$.

If f is a constant, consider the rules for f. If there are no timed rules or rules with the action τ, then vacuosly maximal progress holds for f. Otherwise, let T be the set of rules for f with the action τ, and let R be the set of timed rules for f. Since all rules for f have no active arguments, we deduce by condition (14) that every rule $r \in R$ is below some rule $r' \in T$. Since all rules in T are applicable, we infer that $f \xrightarrow{\tau}$ and no rules in R are ever enabled. Hence, $f \stackrel{\sigma}{\nrightarrow}$ as required.

Consider any rule r and σ-rule r' for f such that active arguments of r and r' are contained in J. Then, condition (14) requires that if the action of r is τ, i.e. r can be used to derive a τ transition, then r is above r'. As a result, any σ-rule for f can only be applied when no rules for f with the action τ are applicable. This is sufficient to ensure that if $f(\boldsymbol{p}) \xrightarrow{\tau}$, then $f(\boldsymbol{p}) \stackrel{\sigma}{\nrightarrow}$. □

Example 23 Consider a timed process language which is obtained by adding $\lfloor\ \rfloor(\ )$ from Example 16 to the time preserving extension of CCS. By Theorem 19, all processes in the new process language are time deterministic. We shall alter this language so it also satisfies maximal progress. We easily check that $\lfloor\ \rfloor(\ )$ satisfies (14). As for the time preserving versions of CCS operators, we put their σ-rules below their τ-rules and other rules with the action τ. So, for the prefixing with τ we have $\tau.X \xrightarrow{\sigma} \tau.X < \tau.X \xrightarrow{\tau} X$. 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 operator, in order to satisfy (14) we put its σ-rule below its synchronisation rule. Thus, the resulting language, henceforth called *TL*, satisfies maximal progress.

For many timed operators condition (14) is too strong. For example, consider a time preserving extension of CCS with the delay operator Δ defined below. The rules for the CCS parallel composition are:

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

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

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

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 condition (14). The ordering satisfies $\sigma_{\{1,2\}} < r_{a\bar{a}}, \tau_1, \tau_2$ and $\sigma_1, \sigma_2 < r_{a\bar{a}}, \tau_1, \tau_2$. Also, by (4.a), we obtain $\sigma_1, \sigma_2 < \sigma_{\{1,2\}}$. When p and q satisfy maximal progress, we do not need the ordering to satisfy $\sigma_{\{1,2\}}, \sigma_1, \sigma_2 < \tau_1, \tau_2$. This is because if $p \xrightarrow{\tau}$ and $q \xrightarrow{\tau}$, then by the maximal progress property, $p \stackrel{\sigma}{\not\rightarrow}$ and $q \stackrel{\sigma}{\not\rightarrow}$ and τ_1 and τ_2 are applicable and enabled, but σ_1 and σ_2 are not applicable. As a result, $p \parallel q \xrightarrow{\tau}$ and $p \parallel q \stackrel{\sigma}{\not\rightarrow}$. Thus, we do not need to rely on $\sigma_{\{1,2\}}, \sigma_1$ and σ_2 to be below the τ-rules τ_1 and τ_2 to disable $\sigma_{\{1,2\}}, \sigma_1$ and σ_2. On the other hand, $\sigma_{\{1,2\}} < r_{a\bar{a}}$ and $\sigma_1, \sigma_2 < r_{a\bar{a}}$ are essential to guarantee the maximal progress property. In fact, the corresponding result is valid for all time preserving operators.

However, for time altering operators, the full strength of condition (14) is required. Let the rules for Δ be as follows:

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

Without the ordering on rules, we have $\Delta p \xrightarrow{\tau}$ and $\Delta p \xrightarrow{\sigma}$ for p that satisfies maximal progress. This is because rule σ is always enabled. The problem is solved by ordering τ_1 and σ according to (14): $\tau_1 < \sigma$.

The maximal progress property can be shown to hold under a slightly weaker condition.

Proposition 24 Let L be any timed process language and T be its LTS. T satisfies the maximal progress property if for every f in L the following holds:

For all rules r and σ-rules r' for time preserving f and all priority levels J for f,

$$\textbf{if } act(r) = \tau \textbf{ and not } tau(r) \textbf{ and } active(r) \cup active(r') \subseteq J \textbf{ then } r' < r \tag{15}$$

For all rules r and σ-rules r' for time altering f and all priority levels J for f,

$$\textbf{if } act(r) = \tau \textbf{ and } active(r) \cup active(r') \subseteq J \textbf{ then } r' < r \tag{16}$$

ET-LOTOS [22, 23] does not satisfy maximal progress as process $\mathbf{i}\{1\};P$, whose syntax is explained in Section 5.6, can both idle for one time unit and perform τ immediately. However, ET-LOTOS satisfies a milder property of 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.

4.2 Urgency versus Patience and Weak Timelock Freeness

A process is urgent if it is able to perform some visible action without any delay or it is completely deadlocked. A transition system is urgent if it has an urgent process. We say that an n-ary operator f is *urgent* if, for some n-ary vector of non-urgent processes $\boldsymbol{p}$, we have $f(\boldsymbol{p}) \stackrel{\tau}{\nrightarrow}$ and $f(\boldsymbol{p}) \stackrel{\sigma}{\nrightarrow}$.

Example 25 Urgent prefixing $\alpha{:}X$ [26, 30] is a time altering urgent operator. Its defining rule is $\alpha{:}X \stackrel{\alpha}{\to} X$ but there are no σ-rules. A completely deadlocked process operator called *timelock* [26, 30], which has no defining rules, is also a time altering urgent operator.

In some situations time deadlock is undesirable. In order to avoid it we shall require the following two conditions.

For all rules r and σ-rules r' for f and all priority levels J for f

$$\textbf{if } \ act(r) \in \mathsf{Vis} \ \textbf{ and } \ active(r) \cup active(r') \subseteq J \ \textbf{ then } \ r' \not< r \tag{17}$$

For each priority levels J for f there exists a σ-rule r for f such that

$$active(r) \subseteq J \tag{18}$$

Condition (17) resembles in some sense (14). For the same r, r' and J as in (14), (17) insists that if the action of r is a visible action, then r cannot be higher in the ordering than the σ-rule r'. Condition (18) says that, for each level of priority for f, there exists at least one σ-rule with active arguments in J.

It can be shown that if a time preserving f satisfies (14) and (17), then, for any non-urgent $\boldsymbol{p}$, we have either $f(\boldsymbol{p}) \stackrel{\tau}{\to}$ or $f(\boldsymbol{p}) \stackrel{\sigma}{\to}$, i.e. f is not urgent. This is roughly because either $f(\boldsymbol{p}) \stackrel{\tau}{\to}$ follows by some r with the action τ (due to (14)) or, if there is no such applicable rule, $f(\boldsymbol{p}) \stackrel{\sigma}{\to}$ follows by one of σ-rules since they are never blocked by any rule with visible actions (due to (17)). Similarly, if a time altering f satisfies (14), (17) and (18) then, for any non-urgent $\boldsymbol{p}$, either $f(\boldsymbol{p}) \stackrel{\tau}{\to}$ or $f(\boldsymbol{p}) \stackrel{\sigma}{\to}$.

With help of Lemma 21 we have the following theorem.

Theorem 26 Let L be any timed process language and T be the LTS generated by L. If all operators in L satisfy (14) and (17) and all time altering operators in L also satisfy (18), then T is *not* urgent and thus it satisfies the patience and weak timelock freeness properties.

Proof. Firstly, we will prove that T is not urgent by structural induction. Consider process term $f(\boldsymbol{p})$ and assume that $\boldsymbol{p}$ is not urgent, i.e. $p_i \stackrel{\tau}{\to}$ or $p_i \stackrel{\sigma}{\to}$ for all i. Let R be the set of all rules that can be used to derive transitions of $f(\boldsymbol{p})$, and let $J \in Levels(f)$ be such that $active(R) \subseteq J$. Then, either there is $r \in R$ with $act(r) = \tau$ or, for all $r \in R$, we have $act(r) \in \mathsf{Vis}$. In the first case, using condition (14) we easily obtain $f(\boldsymbol{p}) \stackrel{\tau}{\to}$ whether f is time preserving or time altering. In the second case, for time preserving f, condition (17) tells us that σ-rule σ_J is not blocked by (i.e. below) any rule in R, thus potentially applicable. For time altering f, condition (18) guarantees that there is a σ-rule with active arguments in J which is potentially applicable. For both kinds of f, we use induction and obtain either $f(\boldsymbol{p}) \stackrel{\sigma}{\to}$ by appropriate σ-rule or $f(\boldsymbol{p}) \stackrel{\tau}{\to}$ by one of the τ-rules.

The patience property holds in T by Lemma 21 because T is not urgent. Moreover, the weak timelock freeness property holds in T because it is a consequence of the patience property. $\square$

Example 27 We show that the language *TL* from Example 23 is not urgent, and thus it satisfies the patience and weak timelock freeness properties. Condition (17) is satisfied because no operator of L has rules with visible actions which are above their σ-rules. Condition (18) holds because $\lfloor\,\rfloor(\,)$ has only one priority level $\{X\}$, $\sigma_\emptyset$ is the σ-rule for $\lfloor\,\rfloor(\,)$ and $active(\sigma_\emptyset) = \emptyset \subseteq \{X\}$.

4.3 Time Persistence

Time persistence means informally that the passage of time does not disable any visible actions: if a process can perform action a, then after any amount of time passage it can still perform a.

Theorem 28 Let L be any timed process language and T be the LTS generated by L. If every time altering operator has either (a) only σ-rules or timed choice rules as its timed rules, or (b) timed rules and no other rules, or (c) no timed rules at all, then T satisfies the time persistence property.

Proof. We prove time persistence by induction on the structure of terms. If f is time preserving and $f(\boldsymbol{p}) \stackrel{\sigma}{\to} f(\boldsymbol{p}')$, then $p_j \stackrel{\sigma}{\to} p'_j$, for all $j \in J$ with $J \in Levels(f)$, and $p'_j = p_j$ otherwise. If $f(\boldsymbol{p}) \stackrel{a}{\to}$ is derivable by rule r, then $p_j \stackrel{\alpha_j}{\to} p''_j$ for all $j \in active(r)$ and some α_i. By induction, $p'_j \stackrel{\alpha_j}{\to}$ holds for each j, so r is applicable to $f(\boldsymbol{p}')$. Hence, $f(\boldsymbol{p}') \stackrel{a}{\to}$.

If f is a time altering and has only σ-rules for its timed rules, then the proof goes as above. If f has only timed rules and no other rules, then $f(\boldsymbol{p})$ can only perform σ. Therefore, $f(\boldsymbol{p}) \stackrel{a}{\nrightarrow}$ and time persistence holds. Finally, if f has no timed rules at all, then $f(\boldsymbol{p}) \stackrel{\sigma}{\nrightarrow}$, and we are done. □

4.4 Timelock Freeness

Another interesting subclass of timed operators are *delay only* operators. A delay only operator f is an operator which is solely defined by σ-rules of the form

$$\frac{\{X_i \stackrel{\sigma}{\to} X'_i\}_{i\in I}}{f(X_1,\dots,X_n) \stackrel{\sigma}{\to} t},$$

where $I \subseteq \{1,\dots,n\}$ and $t = X'_i$, for some $i \in I$, or $t = f(X'_1,\dots,X'_n)$ with $X'_i = X_i$ if $i \notin I$.

Many finite idling operators [49, 19, 26, 36] are essentially delay only operators, assuming that their τ-rules are included.

In the following theorem we will use a slightly stronger version of (17) which contains $act(r) \in \mathsf{Act}$ instead of $act(r) \in \mathsf{Vis}$. This version is denoted by (17') and it says that, for a given priority level J, no rule with active arguments in J can be above any σ-rule with active arguments in J. Hence, no time preserving operators which satisfy (17') can block the passage of time.

Theorem 29 Let L be any timed process language and T be the LTS generated by L. If all time altering operators of L are delay only operators and all time preserving operators of L satisfy (17'), then T satisfies the timelock freeness property.

Proof. We show timelock freeness by induction on the structure of process terms. Assume that f is time preserving and rule r is applicable to derive transitions of $f(\boldsymbol{p})$. Thus, by Definition 13 there is $J \in Levels(f)$ such that $active(r) \subseteq J$. Also, since (17') holds σ_J is potentially applicable. In fact, σ_J is indeed applicable since $p_j \stackrel{\sigma}{\to}$ holds by induction for all $j \in J$. By σ_J we obtain $f(\boldsymbol{p}) \stackrel{\sigma}{\to}$. If f is a time altering delay only operator, then its only defining rules are σ-rules. Thus, $f(\boldsymbol{p}) \stackrel{\sigma}{\to}$ follows by induction.
□

5 Representing Timed Process Languages

In this section we examine six major process languages with time and show how they, or their discrete versions, can be represented in our framework.

5.1 CCS with Time

One of the first extensions of CCS with time was developed by Wang [48, 49]. He used real time domain and defined operational semantics for the extended language. Following this semantics, we can easily define a simple discrete time version Wang's process language within our formalism. 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, thus making it time altering. The passage of time is introduced by a family of delay operators $\varepsilon(d)$, for $d \geq 0$, with the following defining rules:

$$\frac{X \xrightarrow{\sigma} X'}{\varepsilon(d+1).X \xrightarrow{\sigma} \varepsilon(d+1).X'} \qquad \frac{}{\varepsilon(d+1).X \xrightarrow{\sigma} \varepsilon(d).X} \qquad \frac{X \xrightarrow{\chi} X'}{\varepsilon(0).X \xrightarrow{\chi} X'}$$

Delay operators are time altering and, for $d > 0$, they are also examples of delay only operators: their rules satisfy condition (b) of Theorem 28. The rules for operators $\varepsilon(0)$ satisfy condition (a) of Theorem 28.

The remaining operators are time preserving. The choice operator has only one sigma rule, namely $\sigma^{+}_{\{1,2\}}$ from Example 14. 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 SOS rules: we place the communication rule above the sigma rule $\sigma^{\|}_{\{1,2\}}$ from Example 14. As a result maximal progress and time determinacy properties are preserved. Also, as argued above, the persistence property holds by Theorem 28.

Moller and Tofts [26] defined another timed extension of CCS called *Temporal* CCS (TCCS). Unlike in ATP and TPL that we describe below they used a family of delay operators (t): $(t).P$ behaves like P after t time units. These operators are essentially time altering delay only operators. Similarly as in ATP, discussed below, there is urgent prefixing and **0** does not let time pass. Instead of a general delay operator TCCS has two choice operators: a *strong* choice, which is essentially like the time preserving + of CCS, and a *weak* choice $\oplus$. The latter allows to resolve the choice 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 transition system for TCCS is time deterministic, but due to urgent prefixing and $\oplus$ it fails time persistence, timelock freeness and maximal progress. In [27], a version of TCCS called ℓTCCS is proposed. Apart from the mentioned delay operators (which are delay only), ℓTCCS has only time preserving versions of CCS operators. Hence, its transition system satisfies time determinacy and, by Theorems 29 and 28, timelock freeness and time persistence.

5.2 TPL

TPL [19] is a timed extension of CCS with two delay operators: simple delay σX defined by $\sigma X \xrightarrow{\sigma} X$ and the mentioned earlier $\lfloor\ \rfloor(\ )$ (same notation as the operator of ACP).TPL's $\lfloor\ \rfloor(\ )$ is defined by the following rules.

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

Clearly, σX is time altering and delay only. In Example 5 we have shown how $\lfloor\ \rfloor(\ )$ can be redefined as a time altering operator. Prefixing with τ is urgent and time altering since it has no σ-rules. But, the deadlocked operator $\mathbf{0}$, prefixing with visible actions, choice and parallel operators are time preserving. The LTS for TPL is time deterministic and maximal progress holds because of urgent prefixing with τ and the following timed rule for the parallel operator.

$$\frac{X \stackrel{\sigma}{\to} X' \quad Y \stackrel{\sigma}{\to} Y' \quad X \mid Y \stackrel{\tau}{\not\to}}{X \mid Y \stackrel{\sigma}{\to} X' \mid Y'}$$

Also, the patience and weak timelock freeness properties hold for TPL. We have seen in Examples 23 and 27 that TPL can be equivalently expressed as the timed process language *TL*.

5.3 Algebra of Timed Processes

The passage of time in *Algebra of Timed Processes* (ATP) of Nicolin and Sifakis [30] is introduced by the *unit-delay* operator $\lfloor\ \rfloor(\ )$. The operator is defined in our framework by the following SOS rules:

$$\frac{X \stackrel{\alpha}{\to} X'}{\lfloor X \rfloor(Y) \stackrel{\alpha}{\to} X'} \qquad \lfloor X \rfloor(Y) \stackrel{\sigma}{\to} Y$$

We easily see that unit-delay is time altering. ATP has urgent prefixing and urgent $\mathbf{0}$ which are both time altering. However, the choice, parallel and encapsulation operators are time preserving. Furthermore, [30] 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 out framework by families of time altering operators as follows. The family of timeout operators $\cdot \rhd^d \cdot$, where $d > 0$, is defined by the following rules.

$$\frac{X \stackrel{\alpha}{\to} X'}{X \rhd^d Y \stackrel{\alpha}{\to} X'} \qquad \frac{X \stackrel{\sigma}{\to} X'}{X \rhd^{d+1} Y \stackrel{\sigma}{\to} X' \rhd^d Y} \qquad \frac{X \stackrel{\sigma}{\to} X'}{X \rhd^1 Y \stackrel{\sigma}{\to} Y}$$

The family of start delay operators $\lfloor \cdot \rfloor^d \cdot$, where $d > 0$, has the the following rules with the ordering $\sigma_\emptyset^d < \sigma_1^d$ for all $d > 1$.

$$\frac{X \stackrel{\alpha}{\to} X'}{\lfloor X \rfloor^d Y \stackrel{\alpha}{\to} X'}\ r_* \qquad \frac{}{\lfloor X \rfloor^1 Y \stackrel{\sigma}{\to} Y}\ \sigma$$

$$\frac{X \stackrel{\sigma}{\to} X'}{\lfloor X \rfloor^{d+1} Y \stackrel{\sigma}{\to} \lfloor X' \rfloor^d Y}\ \sigma_1^{d+1} \qquad \frac{}{\lfloor X \rfloor^{d+1} Y \stackrel{\sigma}{\to} \lfloor X' \rfloor^d Y}\ \sigma_\emptyset^{d+1}$$

The family of unbounded start delay operators $\lfloor \cdot \rfloor^\omega$ can be defined as follows, with the ordering $\sigma_\emptyset < \sigma_1$.

$$\frac{X \stackrel{\alpha}{\to} X'}{\lfloor X \rfloor^\omega \stackrel{\alpha}{\to} X'}\ r_* \qquad \frac{X \stackrel{\sigma}{\to} X'}{\lfloor X \rfloor^\omega \stackrel{\sigma}{\to} \lfloor X' \rfloor^\omega}\ \sigma_1 \qquad \frac{}{\lfloor X \rfloor^\omega \stackrel{\sigma}{\to} \lfloor X \rfloor^\omega}\ \sigma_\emptyset$$

Finally, the family of execution delay operators $\lceil \cdot \rceil^d \cdot$ ($d > 0$) is defined by the following rules with no ordering.

$$\frac{X \stackrel{\alpha}{\to} X'}{\lceil X \rceil^d Y \stackrel{\alpha}{\to} \lceil X' \rceil^d Y}\ r_* \qquad \frac{X \stackrel{\sigma}{\to} X'}{\lceil X \rceil^1 Y \stackrel{\sigma}{\to} Y}\ \sigma \qquad \frac{X \stackrel{\sigma}{\to} X'}{\lceil X \rceil^{d+1} Y \stackrel{\sigma}{\to} \lceil X' \rceil^d Y}\ \sigma_1$$

A careful reader would have noticed that the start delay and the unbounded start delay operators are not strictly time altering. This is because the orderings on their rules do not satisfy condition (10) as required by part 4 of Definition 15. Despite of this we claim that the above four families of timed operators preserve timed rooted eager bisimulation preorder.

The LTS generated by ATP is time deterministic but not timelock free due to urgent prefixing and **0**. 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 $\lfloor a \rfloor(b) \xrightarrow{\sigma} b \overset{a}{\nrightarrow}$.

5.4 ACP with Discrete Time

Out of many extensions of ACP [9] with discrete time that have been proposed over the years we shall consider ACP_{drt}, ACP *with relative timing*, one of the three different flavours of ACP with discrete time developed by Baeten and Bergstra in [7, 8]. In many respects these three timed versions of ACP supersede a lot of the previous work on discrete time ACP. Additionally, we also look at the recent version ACP with time by Baeten [5].

The operational rules for some ACP operators do not fit directly into the OSOS format, they fit into a more general format, called *panth* [46], that allows predicates and negative premises. However, with some encoding of ACP operators one can give ACP a satisfactory OSOS semantics[1]. Here, we shall concentrate on the method for adding discrete time to ACP.

ACP_{drt} is an extension of ACP with several timed and untimed operators. Since in this process language 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 $\underline{\underline{a}}$ in [8, 5]) 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 with 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 [8]. 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* operator σ_{rel}. It delays its argument by one time unit, and is defined by $\sigma_{rel}(X) \xrightarrow{\sigma} X$ provided the argument is not equal to $\dot{\delta}$. If we encode immediate deadlock by a special action $\dagger$, then σ_{rel} can be defined by the rules

$$\frac{}{\sigma_{rel}(X) \xrightarrow{\sigma} X}\ r_\sigma \qquad\qquad \frac{X \xrightarrow{\dagger} X'}{\sigma_{rel}(X) \xrightarrow{\dagger} X'}\ r_\dagger$$

together with the ordering $r_\sigma < r_\dagger$ and $r_\dagger < r_\dagger$. Note that the second rule disables itself, thus it can never be applied. Because $r_\sigma < r_\dagger$, rule r_σ can only be applied when the premises of $r_\dagger$ are false, namely when $X \overset{\dagger}{\nrightarrow}$.

The choice operator $+$ has the timed rules as in Example 14. For parallel composition there are the merge operator, $\|$, the left merge operator, $\lfloor\!\lfloor$, and the communication operator $|$ with the usual SOS rules. The timed versions of these operators are obtained by adding the following σ-rules:

$$\frac{X \xrightarrow{\sigma} X' \quad Y \xrightarrow{\sigma} Y'}{X \| Y \xrightarrow{\sigma} X' \| Y'} \qquad \frac{X \xrightarrow{\sigma} X' \quad Y \xrightarrow{\sigma} Y'}{X \lfloor\!\lfloor Y \xrightarrow{\sigma} X' \lfloor\!\lfloor Y'} \qquad \frac{X \xrightarrow{\sigma} X' \quad Y \xrightarrow{\sigma} Y'}{X \mid Y \xrightarrow{\sigma} X' \mid Y'}$$

[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.

In ACP [9], the second argument of left-merge is not active. However, in the considered timed version of ACP the second argument of $\|\!\!\lfloor$ is tested in the premises of rules that define immediate deadlock for $\|\!\!\lfloor$. Thus, there is $Y \xrightarrow{\sigma} Y'$ in the premises of the σ-rule for the left-merge.

A somewhat different method for extending ACP with relative discrete time was proposed by Baeten in [5]. The analysis of the interdependence between the notions of action constants, termination, time and silent actions resulted in a redesign of ACP, with execution of actions and termination being separated. Instead of action constants Baeten uses action prefixing, and leaves sequential composition as a basic operator, although it can be eliminated. There are two types of constants for termination: δ stands for deadlock and ε for successful termination. There three variants of these constants: δ and ε allow delay, $\underline{\underline{\delta}}$ and $\underline{\underline{\varepsilon}}$ do not allow time to progress beyond the current time slice, and $\dot{\delta}$ and $\dot{\varepsilon}$ are immediate deadlock and termination respectively, and they allow no delay. There is urgent prefixing $\underline{\underline{a}}$ defined by $\underline{\underline{a}}.X \xrightarrow{a} X$, and ordinary action prefixing with two SOS rules: $a.X \xrightarrow{a} X$ and $a.X \xrightarrow{\sigma} X$. The passage of time is introduced by a single time unit delay operator σ. There is the choice operator $+$ defined as in Example 14, and versions of sequential composition and prefix iteration operators. All the operators are defined operationally by rules in the panth format. Two predicates are employed to define which of the six termination constants represent successful termination or deadlock, and which can pass time.

There are two reasons why the above described process language cannot be represented faithfully as a timed rebo process language. The SOS rules in [5] do not distinguish between silent and visible actions, and since they use negative premises, our conditions concerning silent rules and timed rules cannot be met. Moreover, there are rules for the sequential composition and prefix iteration operators that have +, a non-τ-preserving operator, in the target of their conclusion, thus failing condition 3 of Definition 18. However, we claim that the language can be given an OSOS formulation. We extend the set of action A with special actions σ, $\surd$ and $\dagger$ not in A. $\surd$ denotes the action of terminating successfully and $\dagger$ indicates immediate deadlock. We redefine the mentioned predicates by means of these special actions as follows:

ε :	$\varepsilon \xrightarrow{\sigma} \varepsilon$	$\varepsilon \xrightarrow{\surd} \underline{\underline{\delta}}$	
δ :	$\delta \xrightarrow{\sigma} \delta$		
$\underline{\underline{\varepsilon}}$:		$\underline{\underline{\varepsilon}} \xrightarrow{\surd} \underline{\underline{\delta}}$	
$\underline{\underline{\delta}}$:			
$\dot{\varepsilon}$:		$\dot{\varepsilon} \xrightarrow{\surd} \underline{\underline{\delta}}$	$\dot{\varepsilon} \xrightarrow{\dagger} \underline{\underline{\delta}}$
$\dot{\delta}$:			$\dot{\delta} \xrightarrow{\dagger} \underline{\underline{\delta}}$

Notice, that there no rules for $\underline{\underline{\delta}}$. The choice operator is defined as in Example 14 but without the silent rules, where $a \in A \cup \{\surd\}$, and it has the additional rule

$$\frac{X \xrightarrow{\dagger} X' \quad Y \xrightarrow{\dagger} Y'}{X + Y \xrightarrow{\dagger} Y'}.$$

The sequential composition, denoted by $\cdot$, has the following OSOS rules, where $\chi \in A \cup \{\surd\}$:

$$\frac{X \xrightarrow{a} X'}{X \cdot Y \xrightarrow{a} X' \cdot Y'}\, r_a \qquad \frac{X \xrightarrow{\surd} X' \quad Y \xrightarrow{\chi} Y'}{X \cdot Y \xrightarrow{\chi} Y'}\, r_{\surd\chi}$$

$$\frac{X \xrightarrow{\surd} X'' \quad X \xrightarrow{\sigma} X' \quad Y \xrightarrow{\sigma} Y'}{X \cdot Y \xrightarrow{\sigma} X' \cdot Y \,+\, Y'}\, r_{\sigma\sigma} \qquad \frac{X \xrightarrow{\surd} X'' \quad Y \xrightarrow{\sigma} Y'}{X \cdot Y \xrightarrow{\sigma} Y'}\, r_{\surd\sigma} \qquad \frac{X \xrightarrow{\sigma} X'}{X \cdot Y \xrightarrow{\sigma} X' \cdot Y}\, r_{\sigma}$$

$$\frac{X \xrightarrow{\dagger} X' \quad X \xrightarrow{\surd} X'' \quad Y \xrightarrow{\dagger} Y'}{X \cdot Y \xrightarrow{\dagger} Y'}\, r_{\dagger\dagger} \qquad \frac{X \xrightarrow{\surd} X''}{X \cdot Y \xrightarrow{\surd} X''}\, r \qquad \frac{X \xrightarrow{\dagger} X'}{X \cdot Y \xrightarrow{\dagger} X'}\, r_{\dagger}$$

The ordering is $r_{\surd\sigma} < r_{\sigma\sigma}$, $r_\sigma < r_{\sigma\sigma}$, and $r_\dagger < r$ with $r < r$. Notice that r is a dummy rule that is never enabled, and only serves to produce the effect of $X \overset{\surd}{\not\rightarrow}$ in the premises of $r_\dagger$.

5.5 CSP with Time

The most popular extensions of CSP with time use the dense time domain [34, 36, 37]. 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 [37]; it contains operational semantics of the language and a discussion concerning several timed properties of the generated timed transition system. There are many ways discrete time can be added to CSP. Here, we consider discrete versions of Timed CSP operators that introduce the passage of time and alter the definitions of other operators by adding appropriate timed SOS rules. Our approach is close in spirit to that taken by Roscoe in [35].

The passage of time can be introduced in the language by several means. The deadlocked process $STOP$ has the single timed rule $STOP \xrightarrow{\sigma} STOP$, hence, it is time preserving. 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 sigma rule. We have a family of delay processes $WAITd$ with $d \geq 0$. $WAITd$ is defined as the process $STOP \overset{d}{\triangleright} SKIP$, where $\overset{t}{\triangleright}$ is a member of a family of timeout operators defined as follows:

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

In our terminology the timeout operators are time altering since the timed rules for $\overset{d+1}{\triangleright}$ are not σ-rules, and $\overset{0}{\triangleright}$ has no timed rules at all. The external choice operator has the usual σ-rule that requires synchronisation of time passage of its both arguments. 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. As there is no prefixing with silent actions this property is guaranteed by careful definitions of the three operators that introduce silent actions, namely the internal choice, the hiding and 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 preserves maximal progress. The hiding is defined by the usual rules together with the timed rule

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

In our framework this is defined by a simple timed rule, which in fact is a σ-rule,

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

that is placed below the two standard rules for the hiding operator. This makes the hiding a time altering operator. 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 $\sigma_1 < r_\surd$. We notice that our definitions of the above three operators satisfy condition (14), namely the operators' timed rules are below their relevant τ-rules. Thus, by Theorem 22 maximal progress holds. Also, Timed CSP and this discrete version of it satisfy the time persistence property. In fact, Timed CSP satisfies a stronger property called "constancy of offers" in [37]: if $p \xrightarrow{\sigma} p'$, then ($p \xrightarrow{a}$ if and only if $p' \xrightarrow{a}$).

5.6 ET-LOTOS

ET-LOTOS [22, 23] is an extension of LOTOS that allows the modelling of real-time behaviours. We are not able to model data types of ET-LOTOS and our framework does not support the dense time yet, but we can represent operators of ET-LOTOS for a discrete time domain. Since in our framework we use $\xrightarrow{\sigma}$ transitions and not $\xrightarrow{d}$ transitions, where $d \in \mathbb{N}$ represents a number of time units, we will define ET-LOTOS operators with $\xrightarrow{\sigma}$. The operator **stop** is a time preserving altering operator with a single rule

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

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:

$$\frac{X \xrightarrow{\alpha} X'}{\Delta^0 X \xrightarrow{\alpha} X'} \qquad \frac{}{\Delta^d X \xrightarrow{\sigma} \Delta^{d-1} X}\quad d > 0$$

Observable action prefixing is essentially timed rebo operator although we cannot model directly *satisfaction predicates* of ET-LOTOS. Prefixing with silent action $\mathbf{i}\{d\};P$, where $\{d\}$ is a *life reducer*, models a form of urgency of silent actions: **i** will occur within the next d time units, and $\mathbf{i}\{d\};P$ will not idle for more than d time units. We define $\mathbf{i}\{d\}$ 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: all their defining rules can be easily expressed in our framework. However, since we require that maximal time synchrony also holds, we would also need additional timed rules for the above operators in the fashion of rules σ_1^+ and σ_2^+ for the CCS + in Example 14.

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 a *lookahead* as in [23]. The two traditional rules are, where we call the second rules r_a for each $a \in A$:

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

The required timed rule is simply

$$\frac{X \stackrel{\sigma}{\rightarrow} X'}{\textbf{hide}\ A\ \textbf{in}\ X \stackrel{\sigma}{\rightarrow} \textbf{hide}\ A\ \textbf{in}\ X'}\ \sigma$$

and the ordering is $\sigma < r_a$ for all $a \in A$. It is easy to 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 hide operator is not time altering. This is because the ordering on its rules do not satisfy condition (10) as required by part 4 of Definition 15. Since $\sigma < r_a$ for all $a \in A$, the definition requires $\sigma < \tau_1$ as well. The lack of this ordering makes the operator unsafe with respect to our timed preorder $\underset{\approx}{\sqsubset}_{tr}$. In 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 $P \equiv \tau.\tau.a.\mathbf{0} \underset{\approx}{\sqsubset}_{tr} \tau.a.\mathbf{0} \equiv Q$ but **hide** $\{a\}$ **in** $P \not\underset{\approx}{\sqsubset}_{tr}$ **hide** $\{a\}$ **in** Q. This is because, with **hide** $\{a\}$ **in** $P \stackrel{\tau}{\rightarrow}\stackrel{\sigma}{\rightarrow}$, **hide** $\{a\}$ **in** $Q \stackrel{\tau}{\rightarrow}$ **hide** $\{a\}$ **in** $a.\mathbf{0}$ and since r_a is enabled for **hide** $\{a\}$ **in** $a.\mathbf{0}$ the sigma rule σ is disabled, so **hide** $\{a\}$ **in** $a.\mathbf{0} \stackrel{\sigma}{\not\rightarrow}$.

6 Concluding Remarks

We have proposed a uniform framework for defining timed process languages which are compositional with respect to timed rooted eager bisimulation. As a result most process languages with discrete time can be represented in our framework. Moreover, the generality of our framework allows the user to choose any process operators for her/his favourite timed process language provided that two conditions are satisfied: all operators preserve $\underset{\approx}{\sqsubset}_{tr}$ and the time determinacy property holds. These conditions can be easily verified by checking whether or not the ordered rules for the operators satisfy several syntactical constraints.

In future we hope to generalise our results to an abstract time domain which would cover both discrete and dense time.

Acknowledgments

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 no. 09780270. The second author was supported by the Hori Information Science Promotion Foundation.

References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[16] R.J. van Glabbeek and W.P. Weijland. Branching time and abstraction in bisimulation semantics. In G.X. Ritter, editor, *Information Processing 89*, pages 613–618. Elsevier Science Publishers B. V. (North-Holland), 1989. To appear in JACM.

[17] J.F. Groote. Specification and verification of real time systems in ACP. In L. Logrippo, L.R. Probert, and H. Ural, editors, *Proceedings of 10th International Symposium on Protocol Specification, Testing and Verification*, 1990.

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

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

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

[21] A. Jeffrey. A linear time process algebra. In K.G. Larsen and A. Skou, editors, *Proceedings of CAV'91*, volume 575 of *LNCS*. Springer, 1991.

[22] L. Léonard and G. Leduc. An introduction to ET-LOTOS for the description of time-sensitive systems. *Computer Networks and ISDN Systems*, 29(3):271–292, 1997.

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

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

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

[26] F. Moller and C. Tofts. A temporal calculus of communicating systems. In J.C.M. Baeten and J.W. Klop, editors, *Proceedings of CONCUR'90*, volume 458 of *LNCS*. Springer, 1990.

[27] F. Moller and C. Tofts. Relating processes with respect to speed. In J.C.M. Baeten and J.F. Groote, editors, *Proceedings of CONCUR'91*, volume 527 of *LNCS*. Springer, 1991.

[28] X. Nicollin, J.-L. Richier, J. Sifakis, and J. Voiron. ATP: an algebra for timed processes. In *Proceedings of the IFIP TC2 Working Conference on Programming Concepts and Methods*, 1990.

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

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

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

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

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

[34] G.M. Reed and A.W. Roscoe. A timed model for communicating sequential processes. *Theoretical Computer Science*, 58:249–261, 1988.

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

[36] S. A. Schneider. An operational semantics for timed CSP. *Information and Computation*, 116:193–213, 1995.

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

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

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

[40] I. Ulidowski. Finite axiom systems for testing preorder and De Simone process languages. In M. Wirsing and M. Nivat, editors, *Proceedings of the 5th International Conference on Algebraic Methodology and Software Technology AMAST'96*, volume 1101 of *LNCS*. Springer, 1996.

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

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

[43] I. Ulidowski and I.C.C. Phillips. Ordered SOS rules and process languages for branching and eager bisimulations. *Information and Computation*, 178(1):180–213, 2002. Also available at http://www.mcs.le.ac.uk/∼iulidowski/.

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

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

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

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

[48] Y. Wang. Real-time behaviour of asynchronous agents. In J.C.M. Baeten and J.W. Klop, editors, *Proceedings of CONCUR'90*, volume 458 of *LNCS*. Springer, 1990.

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

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

A Congruence Theorem for Rooted Eager Bisimulation

Before giving the proof of Theorem 11, we introduce an alternative characterization of an eager bisimulation, Proposition 32, which is crucial in proving the theorem. It says that eager bisimilar processes also have matching *action refusal* behaviour. We recall that action refusal can be thought of as an inability to perform an action in a stable state [31]. More precisely, process p refuses action α if p is stable and p cannot perform α, namely $p \stackrel{\tau}{\nrightarrow} \stackrel{\alpha}{\nrightarrow}$. We will also use the following notion.

Definition 30 Let $G = (\Sigma, \mathsf{Act}, R, <)$ be a rebo process language. The notion of a process variable being τ-*preserving* in a context over G is defined inductively as follows.

1. X_i is τ-preserving in X_i, for every variable X_i,
2. X_i is τ-preserving in Σ context $f(t_1,\dots,t_n)$ if, for each $j \in \{1,\dots,n\}$,

$$\text{whenever } X_i \in var(t_j) \text{ and } j \in active(f), \text{ then } \tau_j \in rules(f) \text{ and } X_i \text{ is } \tau\text{-preserving in } t_j.$$

Proposition 31 Let $G = (\Sigma, \mathsf{Act}, R, <)$ 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.

Proposition 32 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.

$$
\begin{array}{llll}
(E.a') & \forall p', \alpha.\, [\, p \xrightarrow{\alpha} p' & \text{implies} & \exists q', q''.\, (q \stackrel{\tau}{\Rightarrow} q' \xrightarrow{\widehat{\alpha}} q'' \text{ and } p'Eq'')] \\
(E.b') & p \stackrel{\tau}{\nrightarrow} & \text{implies} & q\Downarrow \\
(E.c') & \forall q', \alpha.\, [\, p\Downarrow \text{ and } q \xrightarrow{\alpha} q' & \text{implies} & \exists p', p''.\, (p \stackrel{\tau}{\Rightarrow} p' \xrightarrow{\widehat{\alpha}} p'' \text{ and } p''Eq')] \\
(E.a'') & p \stackrel{\tau}{\nrightarrow} & \text{implies} & \forall q'.[q \stackrel{\tau}{\Rightarrow} q' \text{ implies } (pEq' \text{ and} \\
 & & (E.a^\star) & \forall a, p'.(p \xrightarrow{a} p' \text{ implies } \exists q'', q'''.(q' \stackrel{\tau}{\Rightarrow} q'' \xrightarrow{a} q''' \text{ and } p'Eq''')) \\
 & & (E.c^\star) & \forall a, q''.(q' \xrightarrow{a} q'' \text{ implies } \exists p'.(p \xrightarrow{a} p' \text{ and } p'Eq'')))] \\
(E.c'') & p\Downarrow \text{ and } q \stackrel{\tau}{\nrightarrow} & \text{implies} & \forall p'.[p \stackrel{\tau}{\Rightarrow} p' \text{ implies } (p'Eq \text{ and} \\
 & & (E.a^\dagger) & \forall a, p''.(p' \xrightarrow{a} p'' \text{ implies } \exists q'.(q \xrightarrow{a} q' \text{ and } p''Eq')) \\
 & & (E.c^\dagger) & \forall a, q'.(q \xrightarrow{a} q' \text{ implies } \exists p'', p'''.(p' \stackrel{\tau}{\Rightarrow} p'' \xrightarrow{a} p''' \text{ and } p'''Eq')))]
\end{array}
$$

Proof. (If part). We only need to show that $p\Downarrow$ implies $q\Downarrow$. Suppose $p\Downarrow$ and $q\Uparrow$ and derive a contradiction. Since $q\Uparrow$ there exists an infinite sequence from q as follows:

$$q \xrightarrow{\tau} q_1 \xrightarrow{\tau} q_2 \xrightarrow{\tau} \cdots$$

Because $p\Downarrow$, $q \xrightarrow{\tau} q_1$ implies, by $(E.c')$, $p \stackrel{\tau}{\Rightarrow} p' \xrightarrow{\widehat{\tau}} p_1$ and p_1Eq_1, for some p' and p_1. Since p_1 is a τ-derivative of p, $p_1\Downarrow$. Again, by $(E.c')$, $p_1 \stackrel{\tau}{\Rightarrow} p'_1 \xrightarrow{\widehat{\tau}} p_2$ where p_2Eq_2. By repeating this derivation process we can construct the following sequence

$$p \stackrel{\tau}{\Rightarrow}\xrightarrow{\widehat{\tau}} p_1 \stackrel{\tau}{\Rightarrow}\xrightarrow{\widehat{\tau}} p_2 \stackrel{\tau}{\Rightarrow}\xrightarrow{\widehat{\tau}} \cdots,$$

where p_nEq_n for all n. Since $p\Downarrow$, there exists p_k such that $p_l = p_k$, for all $k <= l$, and $p_k \stackrel{\tau}{\nrightarrow}$. But we still have $q_k\Uparrow$. This contradicts $(E.b')$.
(Only-if part). Since $p \stackrel{\tau}{\nrightarrow}$ 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 11, our main congruence theorem. Let $G = (\Sigma, \mathsf{Act}, R, <)$ be a rebo process language and let $\rightarrow$ be the transition relation generated by G, that is as in Definition 7.

We start by defining relations $\mathcal{R}$ and $\mathcal{E}$ over $\mathrm{T}(\Sigma) \times \mathrm{T}(\Sigma)$. $\mathcal{R}$ is the least relation that satisfies

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

Hence, $\mathcal{R}$ is the least congruence relation which contains $\sqsubseteq_r$ and $\mathcal{E}$ is the least congruence relation which contains $\sqsubseteq$. We note that, for $f \in \Sigma$ and appropriate vectors of process terms $\boldsymbol{u}$ and $\boldsymbol{v}$, if $\boldsymbol{u}\mathcal{R}\boldsymbol{v}$, then $f(\boldsymbol{u})\mathcal{R}f(\boldsymbol{v})$. Also, we easily deduce that if $u_i\mathcal{E}v_i$, for all $i \in \mathit{active}(f)$, and $u_i\mathcal{R}v_i$, for all $i \notin \mathit{active}(f)$, then $f(\boldsymbol{u})\mathcal{E}f(\boldsymbol{v})$.

All we need to do 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.

1. If $p\mathcal{R}q$, then p and q satisfy properties $(R.a)$ and $(R.c)$ in Definition 4.
2. If $p\mathcal{E}q$, then p and q satisfy properties $(E.a')$, $(E.b')$, $(E.c')$, $(E.a'')$ and $(E.c'')$ in Proposition 32.

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 in fact prove the following stronger properties.

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

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

Consider $(\mathcal{R}.a)$ first. Assume $f(\boldsymbol{u}) \xrightarrow{\alpha} u'$. This means that there is an action rule or a τ-rule or a silent choice rule that is enabled such that $f(\boldsymbol{u}) \xrightarrow{\alpha} u'$ can be derived by the rule. There are three cases:

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

The proofs of these cases follow below.

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

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

   using a ground substitution ρ defined by $\rho(\boldsymbol{X}) = \boldsymbol{u}$ and $\rho(Y_{ij}) = u_{ij}$ for all i and j. Thus, $f(\boldsymbol{u}) \xrightarrow{\alpha} \rho(E[\boldsymbol{X},\boldsymbol{Y}])$. 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 10, $E[\boldsymbol{X},\boldsymbol{Y}]$ contains only τ-preserving operators.

   According to Definition 10, rebo rules may have implicit copies in the premises and in the target provided that these rules are below the relevant τ-rules. For the above rule r we assume that there are implicit copies in the premises of arguments X_i, where $i \in K$ for some $\emptyset \neq K \subseteq I$. In other words, $K = \{i \mid |J_i| > 1\}$ 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 $u' = E[\boldsymbol{u}^\dagger,\boldsymbol{u}']$.

   (6) 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 8 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, condition (3) or (10) tells us that all silent rules that are associated with the rules in $higher(r)$ are members of $higher(r)$. This implies $\rho(X_j) \stackrel{\tau}{\nrightarrow}$, thus $u_j \stackrel{\tau}{\nrightarrow}$ for all $j \in active(higher(r))$.

Now, we can use the inductive hypothesis for the pairs of $\mathcal{R}$ related elements in $\boldsymbol{u}$ and $\boldsymbol{v}$. $u_i \stackrel{\tau}{\nrightarrow}$, for all $i \in active(higher(r))$, implies $v_i \stackrel{\tau}{\nrightarrow}$. 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 $(R.a)$,

$$v_i' \stackrel{a_{ij}}{\rightarrow} v_{ij} \text{ and } u_{ij} \mathcal{E} v_{ij}. \tag{19}$$

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

$$v_i \stackrel{\tau}{\Rightarrow} v_i' \stackrel{a_{i1}}{\rightarrow} v_{i1} \text{ and } u_{i1} \mathcal{E} v_{i1}. \tag{20}$$

Let $\boldsymbol{v}'$ stand for the sequence $v_1', \ldots, v_n'$ such that $v_k' = v_k$ when $k \notin I \cup active(higher(r))$. Let $\boldsymbol{v}^\dagger$ stand for the sequence of only those v_i' where $i \in L$ or $i \notin I$ (ordered in the corresponding way to $\boldsymbol{u}^\dagger$). 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 as $\boldsymbol{u}'$, we obtain $\boldsymbol{u}' \mathcal{E} \boldsymbol{v}''$. Since $E[\boldsymbol{X}^\dagger, \boldsymbol{Y}]$ contains only τ-preserving operators Proposition 31 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 $f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} C[\boldsymbol{v}'] \stackrel{\alpha}{\rightarrow} v''$ for some $C[\boldsymbol{V}]$ and v''. In fact we claim that $C[\boldsymbol{V}] = f(\boldsymbol{X})$. 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.

Claim 33 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 it applies to $f(\boldsymbol{v}^\star)$. We know that $u_k \stackrel{\tau}{\nrightarrow}$ for all $k \in active(higher(r))$, hence $v_k \stackrel{\tau}{\nrightarrow}$ for all appropriate k by $(R.c)$. Clearly, $v_j^\star = v_j$, for all $j \in active(r')$, and $u_j \mathcal{R} v_j^\star$. Since the 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 fact 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 $v_i \stackrel{\tau}{\nrightarrow}$, for $i \in active(higher(r))$, the τ-rules which may need to be applied belong to the set $\{\tau_i \mid i \in I \setminus active(higher(r))\}$. Note that Claim 33 guarantees that these rules are not disabled by non τ-rules in $higher(r)$. Neither they are 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 described above 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 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. Thus $r < r'$ by (4) or (11). So by the above claim rule r' is in fact a τ-rule, say τ_{m_2}, so $r < \tau_{m_2}$. Hence, $m_2 \in active(higher(r))$ and τ_{m_2} applicable: this contradicts the earlier statement that none of the rules in $higher(r)$ is applicable to $f(\boldsymbol{v}^\star)$.

$f(\boldsymbol{v}') \stackrel{\alpha}{\rightarrow} v''$ is obtained by rule r with the substitution ρ' defined by $\rho'(\boldsymbol{X}) = \boldsymbol{v}'$ and $\rho'(Y_{ij}) = v_{ij}$ for all relevant i and j. This is possible because the premises of r are valid under ρ' in $\rightarrow$, namely $v_i' \stackrel{a_{ij}}{\rightarrow} v_{ij}$ for all i and j. Furthermore, as explained above, none of the rules in $higher(r)$ can be applied to derive transitions of $f(\boldsymbol{v}')$. Lastly, we obtain $v'' = E[\boldsymbol{v}^\dagger, \boldsymbol{v}'']$, and so $u' \mathcal{E} v''$.

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

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

using a ground substitution ρ defined by $\rho(\boldsymbol{X}) = \boldsymbol{u}$ and $\rho(Y_j) = u_i'$. Thus, $(\rho(f(\boldsymbol{X})), \alpha, \rho(Y_i)) \in \rightarrow$ or equivalently $f(\boldsymbol{u}) \xrightarrow{\alpha} u_i'$. 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. $u_i \xrightarrow{\alpha} u_i'$.

Conditions (7) and (8) 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 8 tells us that since r is enabled none of the rules in *higher*(r) is applicable. Condition (10) tells us that all silent rules which are associated with the rules in *higher*(r) are members of *higher*(r). This implies $\rho(X_k) \stackrel{\tau}{\not\rightarrow}$, thus $u_k \stackrel{\tau}{\not\rightarrow}$ for all $k \in active(higher(r))$.

By induction hypothesis, $u_i \xrightarrow{\alpha} u_i'$ implies $v_i \stackrel{\tau}{\Rightarrow} v_i' \xrightarrow{\alpha} v_i''$ and $u_i' \mathcal{E} v_i''$ for some v_i' and v_i''. As in the previous case we show $u_k \stackrel{\tau}{\not\rightarrow}$ implies $v_k \stackrel{\tau}{\not\rightarrow}$ for each $k \in active(higher(r))$.

Finally, we derive $f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} C[\boldsymbol{v}'] \xrightarrow{\alpha} v''$, 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. Condition (7) and our assumptions tell us that τ^i exists. Conditions (12) and (11) 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, $v_i' \xrightarrow{\alpha} v_i''$ so $v'' = v_i''$ and $u' \mathcal{E} v''$. If $v_i \stackrel{\tau}{\not\rightarrow}$, then $C[\boldsymbol{V}]$ is $f(\boldsymbol{X})$, $f(\boldsymbol{v}) \xrightarrow{\alpha} v''$, with $v'' = v_i''$, and $u' \mathcal{E} v''$.

3. Let $f(\boldsymbol{u}) \xrightarrow{\alpha} u'$ 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 premise of r is valid under ρ in $\rightarrow$, i.e. $u_i \xrightarrow{\tau} u_i'$.

As in case 1, since r is applicable none of the rules above r are applicable, so $u_k \stackrel{\tau}{\not\rightarrow}$ for all $k \in active(higher(r))$. By induction hypothesis, $u_i \xrightarrow{\tau} u_i'$ implies $v_i \stackrel{\tau}{\Rightarrow} v_i' \xrightarrow{\tau} v_i''$ and $u_i' \mathcal{E} v_i''$ for some v_i' and v_i''. As in case 1 we have $u_k \stackrel{\tau}{\not\rightarrow}$ implies $v_k \stackrel{\tau}{\not\rightarrow}$ for each $k \in active(higher(r))$.

Next, we show $f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} f(\boldsymbol{v}') \xrightarrow{\tau} f(\boldsymbol{v}'')$ for some $\boldsymbol{v}'$ and $\boldsymbol{v}''$. Since $v_k \stackrel{\tau}{\not\rightarrow}$, for all $k \in active(higher(r))$, we deduce that none of the rules in *higher*(r) is applicable to $f(\boldsymbol{v})$ (otherwise, they would be applicable to $f(\boldsymbol{u})$ thus contradicting the earlier assumption). Using Claim 33 we derive $f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} f(\boldsymbol{v}') \xrightarrow{\tau} f(\boldsymbol{v}'')$ from $v_i \stackrel{\tau}{\Rightarrow} v_i' \xrightarrow{\tau} v_i''$ using r and a suitable ground substitution, where $\boldsymbol{v}'$ and $\boldsymbol{v}''$ are vectors of v_j' and v_j'' respectively such that $v_j' = v_j$ and $v_j'' = v_j'$ for $j \neq i$.

Since $u_j' \mathcal{R} v_j''$, for $j \in active(f) \setminus \{i\}$, implies $u_j' \mathcal{E} v_j''$ we deduce $u_j' \mathcal{E} v_j''$ for all $j \in active(f)$. Moreover, $u_j' \mathcal{R} v_j''$ for all $j \notin active(f)$. Hence, by the definition of $\mathcal{E}$, $f(\boldsymbol{u}') \mathcal{E} f(\boldsymbol{v}'')$ as required.

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

The proof of property $(\mathcal{R}.c)$ is very similar to 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 34 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 $i \in active(r)$.

Proof. Assume $f(\boldsymbol{u})\Downarrow$ and let r be enabled at $f(\boldsymbol{v})$. The proof is by a contradiction: we assume $u_i\Uparrow$, for some $i \in active(r)$, and show that it leads to a contradiction with $f(\boldsymbol{u})\Downarrow$ or with r being enabled at $f(\boldsymbol{v})$. We have two cases: either f has the ith τ-rule τ_i or it has not, in which case condition (7) requires that f has the ith silent choice rule τ^i.

In the first case either τ_i is enabled at $f(\boldsymbol{u})$ or it is not. 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 $j \neq i$. We can show that τ_i is again enabled at $f(\boldsymbol{u}')$ as follows. 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 (3) or (10). If r^* is applicable at $f(\boldsymbol{u}')$, then r^* is applicable to $f(\boldsymbol{u})$ since $\boldsymbol{u}'$ differs from $\boldsymbol{u}$ only in the ith component. Note that r^* being applicable to $f(\boldsymbol{u})$ contradicts the fact that τ_i is enabled at $f(\boldsymbol{u})$. Thus, τ_i is again enabled at $f(\boldsymbol{u}')$. Repeating this argument, we can derive an infinite τ-derivation from $f(\boldsymbol{u})$, hence $f(\boldsymbol{u})\Uparrow$.

When τ_i is not enabled at $f(\boldsymbol{u})$ we proceed as follows. 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 $v_j \stackrel{\tau}{\nrightarrow}$, for $j \in active(r^*)$, since r is enabled at $f(\boldsymbol{v})$. In the following consider $R(r^*)$ the set of all rules such as r^* above. Depending on the form of rules in $R(r^*)$, there are the following three cases.

1. There is an action rule, say r', among the rules in $R(r^*)$. 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 $v_j \stackrel{\tau}{\nrightarrow}$, 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 is a silent choice rule, say τ^j, among the rules in $R(r^*)$. 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 property $(R.a)$. Hence, τ^j applies to $f(\boldsymbol{v})$. This contradicts the facts that r is enabled at $f(\boldsymbol{v})$ and that $r < \tau^j$.

3. None of the rules in $R(r^*)$ is an action rule or a silent choice rule, but one of them is a τ-rule. Let this rule be τ_j, and let $u_j \xrightarrow{\tau} u'_j$. Then, $f(\boldsymbol{u}) \xrightarrow{\tau} f(\boldsymbol{u}')$ where $u'_k = u_k$ for $k \neq j$. We notice that at this point we are in the same situation as in the second paragraph of this proof except that we now have $f(\boldsymbol{u}')$ instead of $f(\boldsymbol{u})$. If τ_i is enabled at $f(\boldsymbol{u}')$, then we are done: $f(\boldsymbol{u})\Downarrow$. Otherwise, there exists $r^{*\prime}$ such that $r^{*\prime}$ enabled at $f(\boldsymbol{u}')$ and $\tau_i < r^{*\prime}$. We have the three cases as above but with $f(\boldsymbol{u}')$ and $R(r^{*\prime})$, where $R(r^{*\prime})$ is the set of rules such as $r^{*\prime}$, instead of $f(\boldsymbol{u})$ and $R(r^*)$. If there is an action rule in $R(r^{*\prime})$, we can derive a contradiction as in case 1 above. If there is a silent choice rule in $R(r^{*\prime})$, we can also derive a contradiction as in case 2 above. Otherwise, it remains to assume that none of the rules that are enabled at $f(\boldsymbol{u}')$ is an action rule or silent choice rule, and that one such rule is a τ-rule, say τ_k. Notice that at this point we are in the same situation as at the beginning of this case except that we deal with $f(\boldsymbol{u}')$ and $R(r^{*\prime})$ instead of $f(\boldsymbol{u})$ and $R(r^*)$. We proceed as follows. We repeatedly apply the above argument, beginning in the second paragraph of the proof, to $f(\boldsymbol{u}')$ and its subsequent derivatives by the appropriate τ-rules. At each step we check if τ_i is enabled and, when τ_i is not enabled, if cases 1 and 2 above apply. If at any step one of these three situations occurs, we easily find a contradiction and thus complete the proof. Otherwise, only τ-rules are enabled at each step (this may happen when an active argument of $f(\boldsymbol{u})$, other than u_i, is divergent). Hence, we construct an infinite sequence of τ moves from $f(\boldsymbol{u})$. This contradicts $f(\boldsymbol{u})\Downarrow$.

So far 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 above, we can show that r^* applies to $f(\boldsymbol{v})$: this contradicts the fact that r is enabled at $f(\boldsymbol{v})$. $\square$

Now, we return to property $(\mathcal{R}.c)$. The property is proved by considering three cases that are very similar to those for $(\mathcal{R}.a)$. In each case, the above claim is used to show that $v_i \stackrel{\mu_{ij}}{\rightarrow} v_{ij}$ implies $u_i \stackrel{\tau}{\Rightarrow} u'_i \stackrel{\mu_{ij}}{\rightarrow} u_{ij}$ for some u'_i and u_{ij} such that $u_{ij}\mathcal{E}v_{ij}$.

STATEMENT 2.

We prove that $\mathcal{E}$ is an eager bisimulation by showing that it satisfies the properties of Proportion 32.

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 $p \sqsubseteq_{\approx} q$, 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 properties originating from $(E.a')$, $(E.b')$ and $(E.c')$

$(\mathcal{E}.a')$ if $f(\boldsymbol{u}) \stackrel{\alpha}{\rightarrow} u'$, then $\exists C[\boldsymbol{V}], D[\boldsymbol{X},\boldsymbol{Y}], \boldsymbol{u}^\dagger, \boldsymbol{u}', \boldsymbol{v}^\dagger, \boldsymbol{v}', \boldsymbol{v}'', v''.(f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} C[\boldsymbol{v}'] \stackrel{\widehat{\alpha}}{\rightarrow} v''$ and $v'' = D[\boldsymbol{v}^\dagger, \boldsymbol{v}'']$ and $u' = D[\boldsymbol{u}^\dagger, \boldsymbol{u}']$ and $u'\mathcal{E}v'')$

$(\mathcal{E}.b')$ if $f(\boldsymbol{u}) \stackrel{\tau}{\nrightarrow}$, then $f(\boldsymbol{v})\Downarrow$

$(\mathcal{E}.c')$ if $f(\boldsymbol{u})\Downarrow$ and $f(\boldsymbol{v}) \stackrel{\alpha}{\rightarrow} v'$, then $\exists C[\boldsymbol{V}], D[\boldsymbol{X},\boldsymbol{Y}], \boldsymbol{v}^\dagger, \boldsymbol{v}', \boldsymbol{u}^\dagger, \boldsymbol{u}', \boldsymbol{u}'', u''.(f(\boldsymbol{u}) \stackrel{\tau}{\Rightarrow} C[\boldsymbol{u}'] \stackrel{\widehat{\alpha}}{\rightarrow} u''$ and $u'' = D[\boldsymbol{u}^\dagger, \boldsymbol{u}'']$ and $v' = D[\boldsymbol{v}^\dagger, \boldsymbol{v}']$ and $u''\mathcal{E}v')$

as well as the properties below that correspond to $(E.a'')$ and $(E.c'')$.

$(\mathcal{E}.a'')$ if $f(\boldsymbol{u}) \stackrel{\tau}{\nrightarrow}$ then $\forall v'.[f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} v'$ implies $\exists \boldsymbol{v}'.\ (v' = f(\boldsymbol{v}')$ and $\boldsymbol{v} \stackrel{\tau}{\Rightarrow} \boldsymbol{v}'$ and $f(\boldsymbol{u})\mathcal{E}f(\boldsymbol{v}')$ and

$(\mathcal{E}.a^\star)$ $\forall a, u'.\ [f(\boldsymbol{u}) \stackrel{a}{\rightarrow} u'$ implies $\exists D[\boldsymbol{X},\boldsymbol{Y}], \boldsymbol{u}^\dagger, \boldsymbol{u}', \boldsymbol{v}^\dagger, \boldsymbol{v}'', \boldsymbol{v}''', v'''.\ (f(\boldsymbol{v}') \stackrel{\tau}{\Rightarrow} f(\boldsymbol{v}'') \stackrel{a}{\rightarrow} v'''$ 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 $u'\mathcal{E}v''')]$ and

$(\mathcal{E}.c^\star)$ $\forall a, v''.\ [f(\boldsymbol{v}') \stackrel{a}{\rightarrow} v''$ implies $\exists D[\boldsymbol{X},\boldsymbol{Y}], \boldsymbol{u}^\dagger, \boldsymbol{u}', \boldsymbol{v}^\dagger, \boldsymbol{v}'', u'.\ (f(\boldsymbol{u}) \stackrel{a}{\rightarrow} u'$ and $v'' = D[\boldsymbol{v}^\dagger, \boldsymbol{v}'']$ and $u' = D[\boldsymbol{u}^\dagger, \boldsymbol{u}']$ and $u'\mathcal{E}v'')])]$

$(\mathcal{E}.c'')$ if $f(\boldsymbol{u})\Downarrow$ and $f(\boldsymbol{v}) \stackrel{\tau}{\nrightarrow}$ then $\forall u'.[f(\boldsymbol{u}) \stackrel{\tau}{\Rightarrow} u'$ implies $\exists \boldsymbol{u}'.\ (u' = f(\boldsymbol{u}')$ and $\boldsymbol{u} \stackrel{\tau}{\Rightarrow} \boldsymbol{u}'$ and $f(\boldsymbol{u}')\mathcal{E}f(\boldsymbol{v})$ and

$(\mathcal{E}.c^\dagger)$ $\forall a, v'.\ [f(\boldsymbol{v}) \stackrel{a}{\rightarrow} v'$ implies $\exists D[\boldsymbol{X},\boldsymbol{Y}], \boldsymbol{v}^\dagger, \boldsymbol{v}', \boldsymbol{u}^\dagger, \boldsymbol{u}'', \boldsymbol{u}''', u'''.\ (f(\boldsymbol{u}') \stackrel{\tau}{\Rightarrow} f(\boldsymbol{u}'') \stackrel{a}{\rightarrow} u'''$ 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 $u'''\mathcal{E}v')]$ and

$(\mathcal{E}.a^\dagger)$ $\forall a, u''.\ [f(\boldsymbol{u}') \stackrel{a}{\rightarrow} u''$ implies $\exists D[\boldsymbol{X},\boldsymbol{Y}], \boldsymbol{v}^\dagger, \boldsymbol{v}', \boldsymbol{u}^\dagger, \boldsymbol{u}'', u'.\ (f(\boldsymbol{v}) \stackrel{a}{\rightarrow} v'$ and $u'' = D[\boldsymbol{u}^\dagger, \boldsymbol{u}'']$ and $v' = D[\boldsymbol{v}^\dagger, \boldsymbol{v}']$ and $u''\mathcal{E}v')])]$

Consider $(\mathcal{E}.a')$ first. Assume $f(\boldsymbol{u}) \stackrel{\alpha}{\to} u'$. 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 $f(\boldsymbol{u}) \stackrel{\alpha}{\to} u'$ is derived by the rule. There are three cases, and they are exactly the same as for $(\mathcal{R}.a)$.

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

The proofs of these cases are as follows.

1. Let $f(\boldsymbol{u}) \stackrel{\alpha}{\to} u'$ be derived by a rule r below:

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

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

The proof begins in the same manner as the proof of case 1 for property $(R.a)$ of Statement 1. If $i \in K$, where $K = \{i \mid |J_i| > 1\}$, condition (6) requires $r < \tau_i$. Since r is enabled at $f(\boldsymbol{u})$ we deduce $u_i \stackrel{\tau}{\not\to}$ for all $i \in K$. Thus, by $(E.b')$ and $(E.a'')$ of the inductive hypothesis, there exists v_i' such that $v_i \stackrel{\tau}{\Rightarrow} v_i' \stackrel{\tau}{\not\to}$ and $u_i \mathcal{E} v_i'$. 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,

$$v_i \stackrel{\tau}{\Rightarrow} v_i' \stackrel{a_{ij}}{\to} v_{ij}' \text{ and } u_{ij}\mathcal{E}v_{ij}'. \tag{21}$$

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

$$v_i \stackrel{\tau}{\Rightarrow} v_i' \stackrel{a_{i1}}{\to} v_{i1}' \text{ and } u_{i1}\mathcal{E}v_{i1}' \tag{22}$$

Let $\boldsymbol{v}'$ stand for the sequence $v_1', \ldots, v_n'$ such that:

$$v_i' = \begin{cases} v_i' \text{ as in (21) and (22)} & \text{for } i \in I \\ v_i' \text{ such that } v_i \stackrel{\tau}{\Rightarrow} v_i' \stackrel{\tau}{\not\to} & \text{for } i \in active(higher(r))\backslash I \\ v_i & \text{otherwise.} \end{cases}$$

Clearly, r applies to $f(\boldsymbol{v}')$. Using a similar argument as in the proof for Claim 35 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 contradicts the assumption that $f(\boldsymbol{u})$ is enabled at r. Therefore, r is enabled at $f(\boldsymbol{v}')$. Moreover, similarly as in case 1 of Statement 1, $u' = E[\boldsymbol{u}^\dagger, \boldsymbol{u}']$ 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 $f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} C[\boldsymbol{v}'] \stackrel{\alpha}{\to} v''$ and $C[\boldsymbol{X}] = f(\boldsymbol{X})$, we need a result similar to Claim 33.

Claim 35 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 it 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. In the following three cases, 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 associated τ-rules. Hence, $u_k \stackrel{\tau}{\nrightarrow}$ for $k \in active(r')$. Since $u_k \mathcal{E} v_k$ and $v_k \stackrel{\tau}{\Rightarrow} v_k^\star$ we deduce $u_k \stackrel{a_{kl}}{\rightarrow} u_{kl}$ 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 $v_i^\star \stackrel{a_i}{\rightarrow} v_{i1}^\star$. By the definition of $\mathcal{E}$ we have $u_i \mathcal{R} v_i$. Since $u_i \stackrel{\tau}{\nrightarrow}$, we obtain $v_i \stackrel{\tau}{\nrightarrow}$ and $v_i (= v_i^\star) \stackrel{a_i}{\rightarrow}$. By $(R.c)$, $u_i \stackrel{a_i}{\rightarrow}$, 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 $v_i^\star \stackrel{\tau}{\rightarrow}$, we deduce $v_i \stackrel{\tau}{\rightarrow}$. Hence, by $(R.c)$, $u_i \stackrel{\tau}{\rightarrow}$, implying that τ^i applies to $f(\boldsymbol{u})$.

□

Lastly, we only need to derive $f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} f(\boldsymbol{v}')$. As before, we use the τ-rules from a possibly larger set $\{\tau_k \mid k \in I \cup active(higher(r))\}$. This is because now (21) holds for v_k. It is easy to see that no other τ-rules can be enabled during the derivation $f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} f(\boldsymbol{v}')$: Consider τ_l with $l \notin I \cup active(higher(r))$ and assume that τ_l is enabled at $f(\boldsymbol{v}^\star)$ where $f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} f(\boldsymbol{v}^\star) \stackrel{\tau}{\Rightarrow} f(\boldsymbol{v}')$. If $\tau_k < \tau_l$, for some $k \in I \cup active(higher(r))$, then $u_l \stackrel{\tau}{\nrightarrow}$. This implies $r < \tau_l$ by condition (4) or (11), thus contradicting the assumption that $l \notin active(higher(r))$. Hence, τ_l cannot be above any τ_k for k as above. Since $f(\boldsymbol{u}) \stackrel{\tau}{\nrightarrow}$ we can only have $\tau_l < \tau_k$, for all k, and $\tau_l < r$. The assumption that τ_l is enabled at $f(\boldsymbol{v}^\star)$ implies $v_i^\star \stackrel{\tau}{\nrightarrow}$ for all $i \in I$. Since $u_i \stackrel{\tau}{\nrightarrow}$ we obtain $v_i^\star \stackrel{a_{ij}}{\rightarrow}$ by $(E.a'')$. So, r is applicable to $f(\boldsymbol{v}^\star)$. This contradicts the assumptions that τ_l is enabled at $f(\boldsymbol{v}^\star)$ and $\tau_l < r$.

Our task is to show that at each stage in the derivation of $f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} f(\boldsymbol{v}')$ some of the τ-rules described above are enabled. Suppose that having reached $f(\boldsymbol{v}^\star)$, i.e. $f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} f(\boldsymbol{v}^\star)$, we wish to use one of the transitions $v_m^\star \stackrel{\tau}{\rightarrow} v_m^{\star\prime}$, for $m \in M \subseteq I \cup active(higher(r))$, 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 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 (4) or (11) depending whether f is τ-preserving of τ-sensitive. So by Claim 35 rule r' is in fact a τ-rule, say τ_{m_2}, so $r < \tau_{m_2}$. Hence, $m_2 \in active(higher(r))$ 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 (4) we obtain $\tau_{m_k} < \tau_{m_k}$. This contradicts (5) or (12).

2. The proof is same as for case 2 for $(R.a)$ of STATEMENT 1 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 from the one for case 3 for $(R.a)$ of STATEMENT 1 as follows. It may happen that some v_j have τ transitions, where $j \in active(higher(r))$. But, since $u_j \stackrel{\tau}{\nrightarrow}$, for all those j, we obtain $v_j \Downarrow$ by the inductive hypothesis. Moreover, if $v_j \stackrel{\tau}{\Rightarrow} v_j' \stackrel{\tau}{\nrightarrow}$, then $u_j \mathcal{E} v_j'$ by $(E.a'')$. Thus, we have $\boldsymbol{v}'$ such that $f(\boldsymbol{u}') \mathcal{E} f(\boldsymbol{v}')$ and $f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} f(\boldsymbol{v}')$ in the similar manner as in case 3 of STATEMENT 1.

The proof of $(\mathcal{E}.c')$ 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 subproperties. Of course, we need a claim corresponding to Claim 35 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 a version of Claim 34 for $\mathcal{E}$.

The following result will be useful in the proofs below of properties $(\mathcal{E}.b')$ and $(\mathcal{E}.a'')$.

Claim 36 If $f(\boldsymbol{u}) \mathcal{E} f(\boldsymbol{v})$, $f(\boldsymbol{u}) \stackrel{\tau}{\nrightarrow}$ and $f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} \stackrel{\tau}{\rightarrow} q^\dagger$, then, for all q' if $f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} q' \stackrel{\tau}{\rightarrow} q'' \stackrel{\tau}{\Rightarrow} q^\dagger$, then $q' \stackrel{\tau}{\rightarrow} q''$ is derived by a τ-rule.

Proof. Assume $f(\boldsymbol{u})\mathcal{E}f(\boldsymbol{v})$, $f(\boldsymbol{u}) \stackrel{\tau}{\nrightarrow}$ and $f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow}\stackrel{\tau}{\rightarrow} q^\dagger$. We need to prove that, for all q', if $f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} q' \stackrel{\tau}{\rightarrow} q'' \stackrel{\tau}{\Rightarrow} q^\dagger$, then $q' \stackrel{\tau}{\rightarrow} q''$ is derived by a τ-rule. The proof is by induction on the sequence of process terms q' as defined above. In the base case $q' = f(\boldsymbol{v})$, and we need to show that if $f(\boldsymbol{v}) \stackrel{\tau}{\rightarrow} q''$, then $f(\boldsymbol{v}) \stackrel{\tau}{\rightarrow} q''$ is derived by a τ-rule. Assume for a contradiction that a rule other than τ-rule can be used to derive $f(\boldsymbol{v}) \stackrel{\tau}{\rightarrow} q''$. There are possibly two such rules, namely an action rule with the action τ and the silent choice rule. In both cases we derive a contradiction in the same manner as for the corresponding cases in the inductive step.

As the inductive hypothesis assume $f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} q' \stackrel{\tau}{\rightarrow} q'' \stackrel{\tau}{\rightarrow} q^\dagger$ and that all individual τ transitions in $f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} q' \stackrel{\tau}{\rightarrow} q''$ are derived by τ-rules. This means that $q'' = f(\boldsymbol{v}')$ and $\boldsymbol{v} \stackrel{\tau}{\Rightarrow} \boldsymbol{v}'$.

In the inductive step we need to show that $f(\boldsymbol{v}') \stackrel{\tau}{\rightarrow} q^\dagger$ is derived by a τ-rule. Assume for a contradiction that a rule other than τ-rule can be used to derive $f(\boldsymbol{v}') \stackrel{\tau}{\rightarrow} q^\dagger$. There are two cases:

Case 1: An action rule r' with $act(r') = \tau$ is enabled at $f(\boldsymbol{v}')$.

The premises of r' are valid for $f(\boldsymbol{v}')$, i.e. $v'_k \stackrel{a_{kl}}{\rightarrow} v'_{kl}$ for all $k \in active(r')$ and appropriate l. We show that $u_k \stackrel{\tau}{\nrightarrow}$ for all $k \in active(r')$: Assume for a contradiction that $u_k \stackrel{\tau}{\rightarrow}$ for some k. If τ_k or τ^k is enabled at $f(\boldsymbol{u})$, then $f(\boldsymbol{u}) \stackrel{\tau}{\rightarrow}$: contradiction. Otherwise, there exists a rule $r^\dagger$ such that $act(r^\dagger) \neq \tau$, $\tau_k < r^\dagger$ or $\tau^k < r^\dagger$ and $r^\dagger$ is enabled at $f(\boldsymbol{u})$. Since $k \in active(r')$, $r' < r^\dagger$ by condition (4) or (11). Hence, $r' < \tau_i$ or $r' < \tau^i$ for $i \in active(r^\dagger)$. This implies $v'_i \stackrel{\tau}{\nrightarrow}$, for all such i, since r' is enabled at $f(\boldsymbol{v}')$. If $r^\dagger$ is an action rule, then the facts $f(\boldsymbol{u}) \stackrel{\tau}{\nrightarrow}$ and $r^\dagger$ is enabled at $f(\boldsymbol{u})$ imply that $u_i \stackrel{\tau}{\nrightarrow}$ for all $i \in active(r^\dagger)$. Moreover, since $f(\boldsymbol{u})\mathcal{E}f(\boldsymbol{v})$ we deduce by the definition of $\mathcal{E}$ $u_i\mathcal{E}v_i$ for all $i \in active(r^\dagger)$. Since $r^\dagger$ applies to $f(\boldsymbol{u})$, it also applies to $f(\boldsymbol{v}')$ by $(E.a^\star)$. If $r^\dagger$ is a choice rule with the ith argument, then $u_i\mathcal{R}v_i$ with $u_i \stackrel{\tau}{\nrightarrow}$. Similarly as for the action rule, we deduce, by $(R.c)$, that $r^\dagger$ applies to $f(\boldsymbol{v}')$. In both cases $r^\dagger$ applies to $f(\boldsymbol{v}')$ contradicts the earlier assumptions of r' being enabled at $f(\boldsymbol{v}')$ and $r' < r^\dagger$.

Having shown that $u_k \stackrel{\tau}{\nrightarrow}$, for all $k \in active(r')$, we apply $(E.c^\star)$ to $u_k\mathcal{E}v_k$. From $v_k \stackrel{\tau}{\Rightarrow} v'_k$ and $v'_k \stackrel{a_{kl}}{\rightarrow} v'_{kl}$ 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 r, which is below r', is enabled at $f(\boldsymbol{u})$. Else, there must exist r'' such that $r' < r''$, $act(r'') \neq \tau$ and r'' is enabled at $f(\boldsymbol{u})$. This implies $r' < \tau_j$ or $r' < \tau^j$, and hence $v^\star_j \stackrel{\tau}{\nrightarrow}$, for $j \in active(r'')$, since r' is enabled at $f(\boldsymbol{v}')$. By $(E.a^\star)$, 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.

Case 2: A silent choice rule τ^k enabled at $f(\boldsymbol{v}')$.

We obtain $u_k \stackrel{\tau}{\rightarrow}$ from $v_k \stackrel{\tau}{\rightarrow}$ and $u_k\mathcal{R}v_k$ by $(R.c)$. To stop τ^k being enabled at $f(\boldsymbol{u})$ there must exist an action rule r'' with $act(r'') \neq \tau$ that applies to $f(\boldsymbol{u})$ and $\tau^k < r''$. By condition (3) or (10), $\tau^k < \tau_l$ or $\tau^k < \tau^l$, for $l \in active(r'')$, implying that $v'_l \stackrel{\tau}{\nrightarrow}$. Hence, we deduce similarly as in the previous case that r'' applies to $f(\boldsymbol{v}')$: contradiction.

□

We showed that only τ-rules can be used to derive $f(\boldsymbol{v}') \stackrel{\tau}{\Rightarrow} q^\dagger$. Therefore, $q^\dagger = f(\boldsymbol{v}'')$ for some $\boldsymbol{v}''$ such that $\boldsymbol{v}' \stackrel{\tau}{\Rightarrow} \boldsymbol{v}''$.

Consider $(\mathcal{E}.b')$. Suppose for a contradiction that $f(\boldsymbol{v})\Uparrow$. Claim 36 tells us that only τ-rules can be used to produce τ transitions of $f(\boldsymbol{v})$ and of all its τ-derivatives. Let $M \subseteq active(f)$ be the set of τ-rules that can be used to produce any τ transition of any τ-derivative of $f(\boldsymbol{v})$. Since $active(f)$ is finite $f(\boldsymbol{v})\Uparrow$ implies $v_m\Uparrow$ for some $m \in M$. If $u_m \stackrel{\tau}{\nrightarrow}$, then $v_m\Uparrow$ contradicts the $(E.b')$ part of the inductive hypothesis. If $u_m \stackrel{\tau}{\rightarrow}$, then since $f(\boldsymbol{u}) \stackrel{\tau}{\nrightarrow}$ the set of rules other than τ_m that are enabled at $f(\boldsymbol{u})$, R, is not empty and τ_m is below all rules in R. Let $f(\boldsymbol{v}^\dagger)$ be the first τ-derivative of $f(\boldsymbol{v})$ at which τ_m is enabled. Then, $v^\dagger_k \stackrel{\tau}{\nrightarrow}$ for all $k \in active(R)$. For each $r' \in R$, since r' applies to $f(\boldsymbol{u})$ it also applies to $f(\boldsymbol{v}^\dagger)$ by the fact that, for all appropriate k, $v_k^\dagger \stackrel{\tau}{\nrightarrow}$ 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.

Next, we show $(\mathcal{E}.a'')$. 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 claim that $u_m \stackrel{\tau}{\nrightarrow}$ for all $m \in M$. Assume for a contradiction that $u_m \stackrel{\tau}{\rightarrow}$ for some $m \in M$. Since $f(\boldsymbol{u}) \stackrel{\tau}{\nrightarrow}$ we deduce that there is a rule $r^\star$ such that $\tau_m < r^\star$ and $r^\star$ is applicable to $f(\boldsymbol{u})$. Hence, $u_i \stackrel{\tau}{\nrightarrow}$ for all $i \in active(r^\star)$. By condition (3) or (10), $\tau_m < \tau_i$ or $\tau_m < \tau^i$ for all $i \in active(r^\star)$. Let $\boldsymbol{v}^\star$ be such that $f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} f(\boldsymbol{v}^\star) \stackrel{\tau}{\rightarrow}\stackrel{\tau}{\Rightarrow} f(\boldsymbol{v}')$, $f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} f(\boldsymbol{v}^\star)$ is by τ-rules different from τ_m and $f(\boldsymbol{v}^\star) \stackrel{\tau}{\rightarrow}$ is by τ_m. Since $\tau_m < \tau_i$ or $\tau_m < \tau^i$, for all appropriate i, we have $v_i \stackrel{\tau}{\nrightarrow}$. Moreover, since $r^\star$ applies to $f(\boldsymbol{u})$ it also applies to $f(\boldsymbol{v}^\star)$ by property $(E.a^\star)$. This contradicts the assumption that τ_m is enabled at $f(\boldsymbol{v}^\star)$. Hence, $u_m \stackrel{\tau}{\nrightarrow}$ 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 $\mathcal{E}$.

We continue with $(\mathcal{E}.a'')$. For $(\mathcal{E}.a^\star)$ there are two types of rules $f(\boldsymbol{u}) \stackrel{a}{\rightarrow} u'$ can be derived by.

1. An action rule with all the associated τ-rules. Let this rule be r as in the beginning of STATEMENT 1. Since $u_i \stackrel{\tau}{\nrightarrow}$ for all $i \in active(r)$, we deduce by the inductive hypothesis that for each v'_i there exists v''_i such that $v'_i \stackrel{\tau}{\Rightarrow} v''_i$ and $v''_i \stackrel{a_{ij}}{\rightarrow} v''_{ij}$. Similarly as in case 1 for $(\mathcal{E}.a)$, there exists the derivation $f(\boldsymbol{v}') \stackrel{\tau}{\Rightarrow} f(\boldsymbol{v}'')$ using τ-rules in $\{\tau_k \mid k \in active(r) \cup active(higher(r))\}$. As argued above, r applies to $f(\boldsymbol{v}'')$. If we assume that there exists r' such that $r < r'$ and r' applies to $f(\boldsymbol{v}'')$, then applying the inductive hypothesis, $(E.c^\star)$, to the premises of r' we can show that r' also applies to $f(\boldsymbol{u})$. This contradicts that r is enabled at $f(\boldsymbol{u})$. Therefore, r is enabled at $f(\boldsymbol{v}'')$, By applying r to $f(\boldsymbol{v}'')$ we find v''' such that $f(\boldsymbol{v}'') \stackrel{a}{\rightarrow} v'''$. Similarly as in case 1 for $(\mathcal{E}.a)$ we show $u'\mathcal{E}v'''$.

2. A choice rule. The proof here is a simpler version of case 2 for $(\mathcal{E}.a)$

Consider $(\mathcal{E}.c^*)$. Assume that $f(\boldsymbol{v}') \stackrel{a}{\rightarrow} v''$ is derived by r as in the beginning of STATEMENT 1. 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 $(E.c^*)$ 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 $u_k \stackrel{\tau}{\nrightarrow}$ and $v_k \stackrel{\tau}{\nrightarrow}$ 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 $\mathcal{E}$.

Finally, consider $(\mathcal{E}.c'')$. Since it is very similar to $(\mathcal{E}.a'')$ its proof follows closely the above proof of $(\mathcal{E}.a'')$. Naturally, we need a version of Claim 36 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.

This, at last, concludes the proof of Theorem 11.

B Congruence Theorem for Timed Rooted Eager Bisimulation

Proof of Theorem. The proof is done by defining the following relations $\mathcal{R}_\sigma$ and $\mathcal{E}_\sigma$ as for Theorem 11. For non-σ-transitions, the properties are proved in the similar way as in theorem 11. We must prove the properties for σ-transitions in all cases, because by adding σ the process language is not rebo.

First, we prove $\mathcal{R}_\sigma$ is a timed rooted eager bisimulation, provided $\mathcal{E}_\sigma$ is a eager bisimulation. Secondly, we prove $\mathcal{E}_\sigma$ is a eager bisimulation.

$\mathcal{R}_\sigma$ is defined to be 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 $\boldsymbol{u}\mathcal{R}\boldsymbol{v}$,

where $C[\boldsymbol{X}]$ is any Σ context and $\boldsymbol{u}$ and $\boldsymbol{v}$ are vectors of the right length of closed Σ terms. Also $\mathcal{E}_\sigma$ is defined as 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 for each i
  - $u_i\mathcal{E}_\sigma v_i$ if X_i is τ-preserving in $C[\boldsymbol{X}]$, and
  - $u_i\mathcal{R}_\sigma v_i$, otherwise.

If $p \sqsubseteq_{tr} q$, then we are done. Suppose $f(\boldsymbol{u})\mathcal{R}_\sigma f(\boldsymbol{v})$ where $\boldsymbol{u}\mathcal{R}_\sigma\boldsymbol{v}$.

$$(\mathcal{R}_\sigma.a1) \quad \text{if } f(\boldsymbol{u}) \xrightarrow{\alpha} u', \text{ then } \exists C[\boldsymbol{V}], D[\boldsymbol{X},\boldsymbol{Y}], \boldsymbol{u}^\dagger, \boldsymbol{u}', \boldsymbol{v}^\dagger, \boldsymbol{v}', \boldsymbol{v}'', v''.(f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} C[\boldsymbol{v}'] \xrightarrow{\alpha} v'' \text{ and } v'' = D[\boldsymbol{v}^\dagger, \boldsymbol{v}''] \text{ and } u' = D[\boldsymbol{u}^\dagger, \boldsymbol{u}'] \text{ and } u'\mathcal{E}_\sigma v'')$$

$$(\mathcal{R}_\sigma.a2) \quad \text{if } f(\boldsymbol{u}) \xrightarrow{\sigma} u', \text{ then } \exists D[\boldsymbol{X},\boldsymbol{Y}], \boldsymbol{u}', \boldsymbol{u}', v'.f(\boldsymbol{v}) \xrightarrow{\sigma} v' \text{ and } v' = D[\boldsymbol{v}, \boldsymbol{v}'] \text{ and } u' = D[\boldsymbol{u}, \boldsymbol{u}'] \text{ and } u'\mathcal{R}_\sigma v')$$

$$(\mathcal{R}_\sigma.c1) \quad \text{if } f(\boldsymbol{u})\Downarrow \text{ and } f(\boldsymbol{v}) \xrightarrow{\alpha} v', \text{ then } \exists C[\boldsymbol{V}], D[\boldsymbol{X},\boldsymbol{Y}], \boldsymbol{v}^\dagger, \boldsymbol{v}', \boldsymbol{u}^\dagger, \boldsymbol{u}', \boldsymbol{u}'', u''.(f(\boldsymbol{u}) \stackrel{\tau}{\Rightarrow} C[\boldsymbol{u}'] \xrightarrow{\alpha} u'' \text{ and } u'' = D[\boldsymbol{u}^\dagger, \boldsymbol{u}''] \text{ and } v' = D[\boldsymbol{v}^\dagger, \boldsymbol{v}'] \text{ and } u''\mathcal{E}_\sigma v')$$

$$(\mathcal{R}_\sigma.c2) \quad \text{if } f(\boldsymbol{u})\Downarrow \text{ and } f(\boldsymbol{v}) \xrightarrow{\sigma} u', \text{ then } \exists D[\boldsymbol{X},\boldsymbol{Y}], \boldsymbol{u}', \boldsymbol{v}', u'.f(\boldsymbol{u}) \xrightarrow{\sigma} u' \text{ and } v' = D[\boldsymbol{v}, \boldsymbol{v}'] \text{ and } u' = D[\boldsymbol{u}, \boldsymbol{u}'] \text{ and } u'\mathcal{R}_\sigma v')$$

As stated above, the properties above are proved in the similar way as $(\mathcal{R}_\sigma.a1)$ and $(\mathcal{R}_\sigma.c1)$.

We consider $(\mathcal{R}_\sigma.a2)$.

If f is time preserving and $f(\boldsymbol{u}) \xrightarrow{\sigma} f(\boldsymbol{u}')$ by σ_I, then $u' = f(\boldsymbol{u}')$ and $D[\boldsymbol{X},\boldsymbol{Y}] = f(\boldsymbol{Y})$ where $u_i \xrightarrow{\sigma} u_i'$ for $i \in I$ and $u_i = u_i'$ for $i \notin I$. By the inductive hypothesis of $(R.a2)$ for u_i, σ_I also applies to $f(\boldsymbol{v})$. Suppose some other rule $r \in \mathit{higher}(\sigma_I)$ applies to $f(\boldsymbol{v})$. Then, there are the following two cases:

1. $r = \sigma_J$. $\sigma_I < \tau(j)$ for all $j \in J$. This follows that $u_j \stackrel{\tau}{\not\rightarrow}$ for all $j \in J$ in order for σ_I to be enabled at $f(\boldsymbol{u})$. Hence, by the inductive hypothesis $u_j \xrightarrow{\sigma}$, hence r applies to $f(\boldsymbol{u})$. This contradicts to the fact that σ_I is enabled at $f(\boldsymbol{u})$.

2. r is not a σ-rule. If r is an action rule, then r applies to $f(\boldsymbol{u})$ since $\sigma_I < \tau(k)$ for all $k \in \mathit{active}(r)$ then all premises of r are valid by $(R_\sigma.c1)$. This contradicts that σ_I is enabled at $f(\boldsymbol{u})$.

   If r is $\tau(k)$, then $v_k \xrightarrow{\tau}$. But $u_k \stackrel{\tau}{\not\rightarrow}$ must be the case, because σ_I is enabled at $f(\boldsymbol{u})$. This contradicts to the assumption of $(R_\sigma.c1)$ for u_k and v_k.

If f is time altering, $f(\boldsymbol{u}) \xrightarrow{\sigma} E[\boldsymbol{u},\boldsymbol{u}']$ by a timed rule r_σ where $u_i \xrightarrow{\sigma} u_i'$ for $i \in \mathit{active}(r)$. Similar to the previous case, r_σ applies to $f(\boldsymbol{v})$.

If there exists $r \in \mathit{higher}(r_\sigma)$ that applies to $f(\boldsymbol{v})$, then r is one of the followings:

1. Let r be a timed rule. Clearly all the premises are also valid for $f(\boldsymbol{u})$ by $(R_\sigma.c2)$ since $u_k \stackrel{\tau}{\not\rightarrow}$ for $k \in \mathit{active}(r)$. This contradicts to that r_σ is enabled at $f(\boldsymbol{u})$.

2. Let r be an action rule. Similar to the previous case, because $u_k \stackrel{\tau}{\not\rightarrow}$, all the premises of r are also valid for $f(\boldsymbol{u})$ by $(R_\sigma.c1)$. Hence r applies to $f(\boldsymbol{u})$, which contradicts to that r_σ is enabled at $f(\boldsymbol{u})$;

3. Let r be $\tau(k)$. Then, $v_k \xrightarrow{\tau}$ but $u_k \not\xrightarrow{\tau}$. This contradicts that $(R_\sigma.a2)$ holds for u_k and v_k.

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

$(R_\sigma.c2)$ is proved in the similar way to $(R.a2)$, where claim 34 also holds when r is a timed rule.

Next we prove $\mathcal{E}_\sigma$ is an eager bisimulation, by showing that the properties of proposition 32 hold not only for a non-σ-transition but also for a σ-transition.

Given $f(\boldsymbol{u})$ and $f(\boldsymbol{v})$ where $u_i\mathcal{E}_\sigma v_i$ for i where τ_i exists, $u_i\mathcal{R}_\sigma v_i$ otherwise, following the STATEMENT 2 in theorem 11 we shall show the following properties:

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

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

$(\mathcal{E}_\sigma.a''_\sigma)$ if $f(\boldsymbol{u}) \not\xrightarrow{\tau}$, then $\forall q'.[f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} q'$ 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 $u'\mathcal{E}_\sigma v''')])]$

$(\mathcal{E}_\sigma.c''_\sigma)$ if $f(\boldsymbol{u})\Downarrow$ and $f(\boldsymbol{v}) \not\xrightarrow{\tau}$, then $\forall p'.[f(\boldsymbol{u}) \stackrel{\tau}{\Rightarrow} p'$ 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

$(\mathcal{E}.c^\dagger_\sigma)$ $\forall u''.\ [f(\boldsymbol{u}') \xrightarrow{\sigma} u''$ 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 $v' = D[\boldsymbol{u}^\dagger,\boldsymbol{u}']$ 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 $u''\mathcal{E}_\sigma v')])]$

First we consider $(\mathcal{E}_\sigma, a_\sigma)$.

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

2. f is time altering and τ-preserving and $f(\boldsymbol{u}) \xrightarrow{\sigma} u'$ is derived by a timed rule r_σ.

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

4. f is time altering and τ-sensitive and $f(\boldsymbol{u}) \xrightarrow{\sigma} u'$ is derived by a timed rule r_σ.

For each case, we show the property holds.

1. $u' = f(\boldsymbol{u}')$ where $u_i \xrightarrow{\sigma} u_i'$ for $i \in I$ and $u_i = u_i'$ for $i \notin I$. By the inductive hypothesis, $v_i \stackrel{\tau}{\Rightarrow} v_i' \xrightarrow{\sigma} v_i''$ and $u_i' \mathcal{E}_\sigma v_i''$ by $(E_\sigma.a)$. Similar to the proof of the first case of $(E.a)$ in theorem 11, $f(\boldsymbol{v}) \stackrel{\tau}{\Rightarrow} f(\boldsymbol{v}')$ and σ_I applies to $f(\boldsymbol{v}')$. Again similar to $(\mathcal{R}_\sigma.a2)$, if some $r \in \mathit{higher}(\sigma_I)$ applies to $f(\boldsymbol{v}')$, then r also applies to $f(\boldsymbol{u})$. But this contradicts that σ_I is enabled at $f(\boldsymbol{u})$. Since f is τ-preserving, $f(\boldsymbol{u}') \mathcal{E}_\sigma f(\boldsymbol{v}')$.

2. Let r_σ be the rule below. This case is similar to the previous case. Since E only contains τ-preserving operators by the definition, $E[\boldsymbol{u}, \boldsymbol{u}'] \mathcal{E}_\sigma E[\boldsymbol{v}, \boldsymbol{v}'']$ holds.

$$\frac{\{X_i \xrightarrow{\sigma} X_i'\}_{i \in I}}{f(\boldsymbol{X}) \xrightarrow{\sigma} E[\boldsymbol{X}, \boldsymbol{X}']}$$

3. Claim 35 is the case when $r = \sigma_I$. Thus, only τ-rules applies to $f(\boldsymbol{u}^\star)$ where $\boldsymbol{u} \stackrel{\tau}{\Rightarrow} \boldsymbol{u}^\star \stackrel{\tau}{\Rightarrow} \boldsymbol{u}'$. Similar to case 1 of STATEMENT 2 of $(\mathcal{E}.a)$ in theorem 11, $f(\boldsymbol{v}) \stackrel{f}{\Rightarrow} (\boldsymbol{v}')$ is the case.

   The only thing left to show is that $f(\boldsymbol{u}') \mathcal{E}_\sigma f(\boldsymbol{v}'')$, where $v_i \stackrel{\tau}{\Rightarrow} v_i' \xrightarrow{\sigma} v_i''$ such that $u_i' \mathcal{E}_\sigma v_i''$ for τ-preserving X_i in $f(\boldsymbol{X})$ and $u_i' \mathcal{R}_\sigma v_i''$ otherwise. For τ-preserving argument X_i, $u_i' \mathcal{E}_\sigma v_i''$ by the inductive hypothesis. Otherwise, $v_i = v_i'$ and by the inductive hypothesis of $(\mathcal{R}_\sigma.a2)$ $u_i' \mathcal{R}_\sigma v_i''$. This follows $f(\boldsymbol{u}') \mathcal{E}_\sigma f(\boldsymbol{v}'')$ by the definition of $\mathcal{E}_\sigma$.

4. Similar to case 3. The only difference is that since only τ-preserving operators allowed in the target of the timed rule, no consideration is necessary for the non-τ-preserving arguments.

$(\mathcal{E}_\sigma.c_\sigma)$ differs from $(\mathcal{E}_\sigma.a_\sigma)$ in that we have to consider $u_i\Downarrow$ when $f(\boldsymbol{u})\Downarrow$. But since σ transition does not affect this fact, claim 34 in theorem 11 holds also for timed rules.

Next we consider $(\mathcal{E}_\sigma.a_\sigma'')$. We have to consider $(\mathcal{E}_\sigma.a_\sigma^\star)$ and $(\mathcal{E}_\sigma.c_\sigma^\star)$.

$(\mathcal{E}_\sigma.a_\sigma^\star)$ The structure of the proof is similar to $(\mathcal{E}.a'')$. We also have the four cases as $(\mathcal{E}_\sigma.a_\sigma)$. It is also shown that only τ-rules applies to $f(\boldsymbol{u}')$ where $f(\boldsymbol{u}) \stackrel{\tau}{\Rightarrow} f(\boldsymbol{u}')$. Hence, the same timed rule applies to some $f(\boldsymbol{u}''$ such that $f(\boldsymbol{u}') \stackrel{\tau}{\Rightarrow} f(\boldsymbol{u}'')$. It is also similarly shown that the timed rule is actually enabled at $f(\boldsymbol{u}'')$.

$(\mathcal{E}_\sigma.c_\sigma^\star)$ Similar to $(\mathcal{E}.c^\star)$.

$(\mathcal{E}_\sigma.c_\sigma'')$ is similar to $(\mathcal{E}_\sigma.a_\sigma'')$.

□