Electronic Notes in Theoretical Computer Science 70 No. 3 (2002)
URL: http://www.elsevier.nl/locate/entcs/volume70.html 21 pages

Refinement of Actions in a Real-Time Process Algebra with a True Concurrency Model

Harald Fecher, Mila Majster-Cederbaum and Jinzhao Wu

Universität Mannheim
Fakultät für Mathematik und Informatik
D7, 27, 68131 Mannheim, Germany
{hfecher,mcb,wu}@pi2.informatik.uni-mannheim.de

Abstract

In this paper, we develop techniques of action refinement in a real-time process algebra that allows urgent interactions to model timeout. Semantic counterpart is carried out in a real-time non-interleaving causality based setting, timed bundle event structures. We show that our refinement notions have the following nice properties: the observable behaviour of the refined system can be inferred compositionally from the observable behaviour of the original system and from the observable behaviour of the processes substituted for the actions; the timed extensions of observational pomset trace equivalence and observational history preserving bisimulation equivalence are both congruences under our refinement; and the syntactic and semantic refinements coincide up to the aforementioned equivalence relations with respect to a cpo-based denotational semantics.

Key words: action refinement, timed event structure, timed process algebra, true concurrency.

1 Introduction

We consider the design of real-time concurrent systems in the framework of approaches where the basic building blocks are actions. By an action we understand here any activity which is considered as a conceptual entity on a chosen level of abstraction. This allows the development of systems in a hierarchical way, i.e. actions on a higher level are interpreted by more complicated processes on a lower level. Such a change in the level of abstraction is usually referred to as *action refinement*, which is a core operation in the methodology of top-down design of systems, real-time or not.

Action refinement for classical concurrent systems without timing constraints has been thoroughly discussed in the literature [1,8]. The models of concurrency can roughly be distinguished in two groups: interleaving based models, in which the independent execution of two processes is modelled by

specifying the possible interleaving of their actions; and true concurrency models, in which the causal relations between actions are represented explicitly.

A number of timed process calculi have been proposed based on the interleaving approach [20]. Normally, without further restriction or relaxation on actions or refinement operations, in the interleaving approach most of the equivalence relations are not preserved under refinement [7]. These equivalence relations, however, are often used to establish the correctness of the implementation with respect to the specification of concurrent systems. This problem can be overcome by moving to true concurrency models. Also, in the system design phase the local causal dependencies between actions and their timing constraints are important. Interleaving with actions of other parts of the system burdens the design. (Timed) partial order models are considered to be much more appropriate here. Moreover, a causality based model does not suffer from the state explosion problem. Parallelismleads to the sum of the components, rather than to their product as in the interleaving approach.

We study refinement of actions in real-time true concurrent systems. Our aim is to achieve higher reliability with respect not only to functional but also to temporal correctness. In our setting, real-time concurrent systems are described in a timed LOTOS-based process algebra [4] like that proposed in [12,14], where a powerful operation to deal with urgent interactions is incorporated. It is indeed a synthesis of CCS [18] and CSP [11]. We use timed bundle event structures [6,12,13,14] as the semantic model. Please note that bundle event structures have been shown to adequately deal with e.g. parallel composition, and the method to attach timing information does not complicate the theory and it is very powerful [12,15]. We are unaware of any other proposal that incorporates time and timeout in a partial-order setting. However, most protocols are based on timeout mechanisms that are essential for the safety of their behaviour.

There are essentially two interpretations of action refinement, one of which is called syntactic and the other semantic. In syntactic action refinement, an action that occurs in an expression is replaced by a more complicated expression, yielding a more detailed process description [1,8]. Due to its definitional clarity, syntactic action refinement can be easily used without too much insight on semantics. Semantic action refinement is carried out in the semantic domain. It may avoid a confusion of the abstraction levels, which can happen in syntactic refinement. Such a confusion may result in an undesirable situation [7,8].

We adopt the methodology that refinement is modelled as an operator. After defining syntactic and semantic action refinement for our real-time process algebra, we discuss three common issues of interest: *coincidenœ*, *safety* and *congruenœ* problems.

A refinement operation should be safe. That is, the behaviour of the refined system should be the refinement of the behaviour of the original system (in some sense), and vice versa. On the other hand, equivalence relations

are often used to capture when two processes are considered to exhibit the same behaviour. To examine the well-definedness of the refinement operator, we have to try to find such equivalence relations which are congruences under it. The result that syntactic and semantic refinements coincide can give a clear understanding of the concept of action refinement, and meanwhile has important applications in system verifications [16]: the refined syntactic specification can be modelled by semantic action refinement, and the refined semantic model can be specified by syntactic action refinement.

The main contributions of this paper are:

- the notions of syntactic and semantic action refinement operators for the timed process algebra;
- the coincidence result of semantic refinement with syntactic refinement;
- the verification of safety of refinement;
- the congruence results about observational pomset trace and history preserving bisimulation equivalences.

This paper is a further development of [7,17,19]. In [7] action refinement in untimed even tstructures is studied. In [19] a timed version of prime event structures with a simple notion of refinement is defined. Whereas in [17] action refinement in real-time concurrent systems without multi-way synchronization is investigated.

2 A real-time process algebra

In this section, we fix the process algebraic framework that is used to develop real-time concurrent systems, and we present a cpo-based denotational semantics.

2.1 Syntax

Assume a given set Θ of observable actions and an invisible internal silent action τ ($\tau \notin \Theta$). Action $\surd$ ($\surd \notin \Theta \cup \{\tau\}$) indicates the successful termination of a process. Let $\Omega = \Theta \cup \{\tau, \surd\}$. Unlike [5,6,12,13,14], actions are not instantaneous in this paper. They are viewed as compound happenings having durations. Let function $k : Act \to \mathbb{R}^+$ with $k(\tau) = k(\surd) = 0$ assign durations to actions. Here $\mathbb{R}^+ = [0, \infty)$, the set of non-negative reals, denotes the time domain.

Please note that it is reasonable to define $k(\tau) = 0$. A time interval in which the system is silent can be imagined as the system run is separated by a starting and an ending τ-event.

Definition 2.1.1 (*Expression*) The set of (timed) expressions is generated by the following grammar:

$$P ::= 0 \mid 1_T \mid a_T.P \mid P;P \mid P + P \mid P \parallel_A P \mid P\backslash A \mid P[\lambda] \mid P \rhd_t P \mid x \mid \mu x.P.$$

Here, $a \in \Theta \cup \{\tau\}$, $A \subseteq \Theta$, λ: $\Omega \to \Omega$ with $\lambda(\tau) = \tau$, $\lambda(\surd) = \surd$, and $\lambda(a) \neq \surd$ for $a \in \Theta$, is called a relabelling function, V is a set of process variables with $x \in V$, and $T \subseteq \mathbb{R}^+$ and $t \in \mathbb{R}^+$.

In what follows, we assume $dom(\lambda) = \{a \in \Omega \mid \lambda(a) \neq a\}$, representing the set of actions that are really relabelled by λ. We also assume that an action a can only be relabelled by an action with the same duration, i.e., $k(\lambda(a)) = k(a)$. The operators have the following intuitive meaning:
0 denotes inaction. 1_T represents the process that terminates successfully at some time instant in T. In the timed process algebra defined in [12,13,14], the time attachments of successful termination processes are always $[0, \infty)$. In this paper, we need to label them with an arbitrary time instant set T, since actions are no longer instantaneous, and so the time instants at which an action may finish have to be specified.
$a_T.P$ denotes the prefix of action a before P where a finishes at a certain time point in T. $P[\lambda]$ denotes the relabelling of P according to λ. $P\backslash A$ behaves as P, except that the actions in A are abstracted, i.e., turned into τ-actions.
$P_1; P_2$ denotes the sequential composition of P_1 and P_2, where the control is passed to P_2 by the successful termination of P_1. $P_1 + P_2$ indicates the choice between the behaviours described by P_1 and P_2. $P_1 \parallel_A P_2$ denotes the parallel composition of P_1 and P_2, where P_1 and P_2 must perform any actions in $A \cup \{\surd\}$ simultaneously, while the other actions are executed independently from each other. $P_1 \rhd_t P_2$ behaves like P_1, except that if P_1 does not begin to perform an action before time point t, a timeout occurs at t and control is passed to P_2.

Recursive behaviour is described by $\mu x.P$. It can be understood through $x := P$, where $x \in V$ may occur in the body P.

By TPA we denote the set of all the expressions of our language. Furthermore, let $A_P \subseteq \Theta$ be the set of all observable actions occurring in expression P. The time labels of actions consisting of $[0, \infty)$ are usually omitted in an expression.

Example 2.1.2 The following P and P_c are expressions.

$$(1)\ P = a_{(0,3]}.c_{[2,5]}.0 \parallel_{\{c\}} (b_{(0,3]}.0 + c_{(0,8]}.0);$$

$$(2)\ P_c = ((d_{1(0,4]}.1 \rhd_2 d_2.1); 1) \parallel_\emptyset 1_{\{4\}}.$$

2.2 Semantics

In this section, we extend timed bundle event structures (See, e.g. [12]) with action durations and use them as the semantic model of our real-time process algebra.

2.2.1 Timed event structures

A bundle event structure consists of events labelled with actions and two relations between events for causality and for conflict. To say that an event occurs means that the action labelled to it occurs. The symmetric conflict relation, denoted $\sharp$, is a binary relation between events, and the intended meaning of $e \sharp e'$ is that events e and e' cannot both occur in a single system run. Causality is represented by a binary bundle relation, denoted $\mapsto$. It relates sets of events, where all events have to be pairwise in conflict, and events. $X \mapsto e$ means that if event e happens in a system run, exactly one event in set X has happened before and caused e. X is called a *bundle-set*. Event and bundle delay functions $\mathcal{D}$ and $\mathcal{R}$ are used to associate sets of time instants to events and bundles, respectively. $\mathcal{D}(e) = T$ means that event e may finish at some time instant in T. The interpretation of a bundle $X \mapsto e$ with $\mathcal{R}(X, e) = T$ is that if an event in X has finished at certain time point then e is enabled and may finish after some time units in T. In order to specify timeout mechanisms, we use a set $\mathcal{U}$ of urgent events. Such events are forced to occur as soon as they are enabled.

Definition 2.2.1 (*Timed bundle event structure, tes* for short) A tes $\mathcal{E}$ is a tuple $(E, \sharp, \mapsto, l, \mathcal{D}, \mathcal{R}, \mathcal{U})$ with

E, a set of events;
$\sharp \subseteq E \times E$, the irreflexive and symmetric conflict relation;
$\mapsto \subseteq \mathcal{P}(E) \times E$, the bundle relation;
$l : E \to \Omega$, the action labelling function;
$\mathcal{D} : E \to \mathcal{P}(\mathbb{R}^+)$, the event delay function;
$\mathcal{R} : \mapsto \to \mathcal{P}(\mathbb{R}^+)$, the bundle delay function; and
$\mathcal{U} \subseteq \{e \in E \mid l(e) = \tau\}$, the set of urgent events,
such that for any bundle-set X, $e \in \mathcal{U}$ and $e' \in E$,
(1) $(X \times X) \setminus Id_E \subseteq \sharp$;
(2) $(X \mapsto e) \wedge (e \sharp e') \Rightarrow (X \mapsto e') \vee (X \sharp e')$;
(3) there is $t \in \mathbb{R}^+$: $(\mathcal{D}(e) \subseteq \{t\}) \vee (\exists X : (X \mapsto e) \wedge (\mathcal{R}(X, e) \subseteq \{t\}))$.

Here $\mathcal{P}(\cdot)$ denotes the power-set function. $X \sharp e'$ denotes $(\forall e'' \in X : e' \sharp e'')$, and $Id_E = \{(e, e) \mid e \in E\}$.

Constraint (1) enables us to uniquely define a causal ordering between the events in a system run. Constraint (2) is needed to locally decide whether an event can occur by only considering its direct causal predecessors and conflicts. The motivation for Constraint (3) is that urgent events are used for the sole purpose of modelling timeouts which are internal actions of a process and typically can appear at a single time instant only. Please note that an untimed event structure may be considered as a tes in which the durations of actions and bundle and event delays are all zero.

A tes is depicted as follows: Urgent even ts are denoted by open dots, other events by closed dots. Near the dot the action label is given. $e \sharp e'$ is indicated by a dotted line between e and e'. A bundle $X \mapsto e$ is indicated by an arrow to e and to this arrow each ev en t in X is connected via a line. Even t and bundle delays are depicted near to the event and bundle, respectively. Delays $[0, \infty)$ are omitted. If no confusion arises, we identify an event with its action label.

Example 2.2.2 The following $\mathcal{E}$ (left-hand) and $\mathcal{E}_c$ (right-hand) are both tes's. In the sequel, suppose $a =$*read_message*, $b =$*revise_message*, $c =$*send_message*, $d_1 =$*channel_1* and $d_2 =$*channel_2*. We also suppose that $k(a) = 2, k(b) = 2, k(c) = 4$, $k(d_1) = 1$ and $k(d_2) = 0.5$. *Timeout* and *termination* are indicated by urgent in ternal action τ and action $\surd$, respectively .

By TES we represent the set of all tes's. Elements of TES are denoted by $\mathcal{E}, \mathcal{E}_i$, where $\mathcal{E} = (E, \sharp, \mapsto, l, \mathcal{D}, \mathcal{R}, \mathcal{U}), \mathcal{E}_i = (E_i, \sharp_i, \mapsto_i, l_i, \mathcal{D}_i, \mathcal{R}_i, \mathcal{U}_i)$. When necessary, we also use $E_\mathcal{E}$, $\sharp_\mathcal{E}$, $\mapsto_\mathcal{E}$, $l_\mathcal{E}$, $\mathcal{D}_\mathcal{E}$, $\mathcal{R}_\mathcal{E}$ and $\mathcal{U}_\mathcal{E}$ to represent the components of $\mathcal{E}$. For two different tes's $\mathcal{E}_f$ and $\mathcal{E}_g$, we always assume without loss of generality that $E_f \cap E_g = \emptyset$.

We use $init(\mathcal{E})$ to denote the set of initial even tsof $\mathcal{E}$, and $exit(\mathcal{E})$ denotes its set of successful termination events:

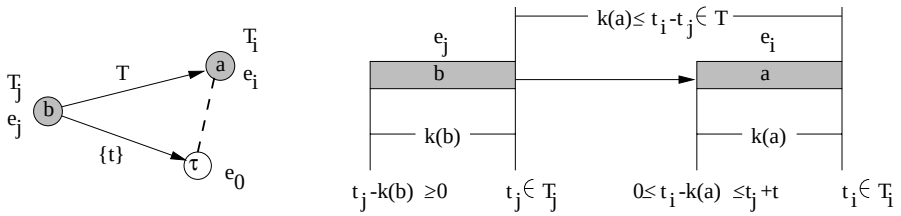
$$init(\mathcal{E}) = \{e \in E \mid \neg(\exists X \subseteq E : X \mapsto e)\}, exit(\mathcal{E}) = \{e \in E \mid l(e) = \surd\}.$$

T odescribe system runs of a tes, we first consider its underlying set of ev en ts combined with time points. A *timed event (of $\mathcal{E}$)* is a pair (e, t), where $e \in E$, $t \in \mathbb{R}^+$. Here t indicates the time instant at which ev en t e finishes. Now suppose that γ is a finite sequence $(e_1, t_1), \cdots, (e_n, t_n)$ of timed events of $\mathcal{E}$, where e_i and e_j are distinct whenever $i \neq j$. By $E(\gamma)$ we denote the set $\{e_1, \cdots, e_n\}$ of events, and $Time_\gamma$ delivers the accompanying time instant of e_i, i.e. $Time_\gamma(e_i) = t_i$. $E(\gamma)$ and $Time_\gamma$ can be similarly defined when γ is a set of distinct timed ev ents. F urthermore, $e \in E(\gamma)$ is said to be *maximal* in γ or $E(\gamma)$ if for any bundle-set X and $e' \in E$, if $e \in X$ and $X \mapsto e'$ then $e' \notin E(\gamma)$. Sequence $\gamma_{i-1} = (e_1, t_1), \cdots, (e_{i-1}, t_{i-1})$ represents the $(i-1)$-th prefix of γ. F or $t \in \mathbb{R}^+$ and $T \subseteq \mathbb{R}^+$, $T - t$ denotes the set $\{t_i - t \mid t_i \in T\} \cap \mathbb{R}^+$ and $T + t$ is defined similarly. We sa y $t \leq T$ if $T = \emptyset$ or t is not greater than each element of T.

In a system run (i) any two events that occur should not be in conflict, and (ii) if a non-initial event happens in a system run then the events that cause it should have happened earlier in this system run. Let

$$en(\gamma_{i-1}) = \{e \in E \setminus E(\gamma_{i-1}) \mid (\forall e_j \in E(\gamma_{i-1}) : \neg(e \sharp e_j)) \wedge (\forall X \mapsto e : X \cap E(\gamma_{i-1}) \neq \emptyset)\}.$$

It is the set of events that are enabled after the performance of γ_{i-1}.



The tes $\mathcal{E}$ depicted above is used to illustrate our basic philosophy about time requirements for system runs. The fact that $(e_j, t_j), (e_i, t_i)$ is a run of $\mathcal{E}$ contains the information shown in the right-hand side of the graph above.

Stated in words, (iii) the time instant t_i accompanying event e_i is required to be in the time instant set labelled to e_i. It is the time instant at which e_i finishes. Therefore, $t_i \in T_i$, and it cannot be smaller than the execution time of e_i, which is understood as the duration of action a labelled to e_i, i.e., $t_i \geq k(a)$. $t_i - k(a)$ is thus the time instant at which e_i begins to occur. Similarly, for t_j we require that $t_j \in T_j$, and $t_j \geq k(b)$. $t_i - k(b)$ is the time instant at which e_j begins to occur; (iv) $t_i - t_j$ is the exhausted time of the system run from the finish of e_j to the finish of e_i. It is required to be in the time instant set attached to the corresponding bundle. So $t_i - t_j \in T$ and it cannot be smaller than the execution time of e_i, i.e, $t_i - t_j \geq k(a)$; (v) Urgent event e_0 did not disable e_i. This implies that the starting time instant of e_i must not be greater than the time instant at which e_0 may occur. The latter, according to our explanation, is $t_j + t$. Hence $t_i - k(a) \leq t_j + t$.

Bearing this in mind, we see that if an event e is enabled after γ_{i-1}, then

$$tm_{\gamma_{i-1}}(e) = [k(l(e)), \infty) \cap \mathcal{D}(e) \cap (\cap_{X \mapsto e, e_j \in X}(t_j + [k(l(e)), \infty) \cap \mathcal{R}(X, e)))$$

consists of all the potential time instants at which event e may finish.

Definition 2.2.3 (*Configuration*) If there exists a timed event sequence $\gamma = (e_1, t_1), \cdots, (e_n, t_n)$ satisfying the conditions

(1) $e_i \in en(\gamma_{i-1})$;
(2) $t_i \in tm_{\gamma_{i-1}}(e_i)$; and
(3) $\forall e \in en(\gamma_{i-1}) \cap \mathcal{U} : (e_i \sharp e) \Rightarrow (t_i - k(l(e_i)) \leq tm_{\gamma_{i-1}}(e))$,

for $1 \leq i \leq n$, then the set of all the timed events occurring in γ is called a (timed) configuration of $\mathcal{E}$.

The third constraint ensures that in the graph above even te_i must start to happen before the time instant at which urgent event e_0 is enabled. Remark that the configurations of a tes depend heavily on the duration function of actions.

We say that a configuration σ of $\mathcal{E}$ *successfully terminates* if there exists $e \in E(\sigma)$ such that e is labelled with the successful termination action $\surd$. $\mathcal{E}$ is said to be *non-forgetful* if for each of its configurations σ that successfully terminates, $E(\sigma)$ contains at least one event which is labelled with an observable action. $\mathcal{E}$ is called *well-labelled* if $E(\sigma) \cap exit(\mathcal{E})$ is empty or a singleton whenever σ is a configuration of it. Let

$$maxt(\mathcal{E}) = \{t \in \mathbb{R}^+ \mid \text{ there exist configuration } \sigma \text{ and } (e,t) \in \sigma : l(e) = \surd\}.$$

It consists of all the time instants at which a successful termination run of the system finishes.

We are interested in the observable behaviour of a system, i.e. we abstract from internal actions. As a consequence, we may especially remove τ-events that are produced by sequential composition and by action refinement, which is defined later.

Definition 2.2.4 (*Observational configuration*) Let σ be a configuration of $\mathcal{E}$. $\{(e,t) \in \sigma \mid l(e) \neq \tau\}$ is an observational configuration of $\mathcal{E}$.

The observational configuration in Definition 2.2.4 is said to be derived from configuration σ. It *successfully terminates*, if σ successfully terminates. By $OC(\mathcal{E})$ we denote the set of all observational configurations of $\mathcal{E}$.

A system run may be represented as an equivalence class of labelled partial orders (pomsets). We define in the following a linear-time equivalence, termed observational pomset trace equivalence, on tes's. A branching-time equivalence, termed observational history preserving bisimulation equivalence (see e.g. [7]), can be defined similarly. This equivalence further records where choices are made. Here we only describe the former equivalence in detail.

Assume that σ_0 is a configuration of $\mathcal{E}$ and $e_i, e_j \in E(\sigma_0)$. By $e_j \mapsto e_i$ we mean that there exists a bundle-set X such that $e_j \in X$ and $X \mapsto e_i$. For the observational configuration σ derived from σ_0, a binary relation $\rightarrow|_\sigma$ on $E(\sigma)$ is defined as the above relation $\mapsto$ in which the causally necessary internal τ-events that occurs are neglected, i.e. $e_j \rightarrow|_\sigma e_i$ (or $\rightarrow_f|_\sigma$ for $\mathcal{E}_f$) iff $e_j \mapsto e_i$ or there exist $e_{k_p} \in E(\sigma)$ with $l(e_{k_p}) = \tau$ for $p = 1, \cdots, m$, such that $e_j \mapsto e_{k_1}$, $e_{k_q} \mapsto e_{k_{q+1}} (q = 1, \cdots, m-1)$ and $e_{k_m} \mapsto e_i$. It is in fact the causality relation that is observable in σ_0. In the sequel, we use $\rightarrow|_\sigma$ again to represent the

transitive closure of $\rightarrow|_\sigma$, and we use $l|_\sigma$ as the restriction of l on $E(\sigma)$, i.e. $l|_\sigma(e) = l(e)$ for $e \in E(\sigma)$.

An observational configuration σ_1 of $\mathcal{E}_1$ and an observational configuration σ_2 of $\mathcal{E}_2$ are said to be *isomorphic*, denoted $\sigma_1 \approx \sigma_2$, if there exists a bijection $h : E(\sigma_1) \rightarrow E(\sigma_2)$, such that for arbitrary $e, e' \in E(\sigma_1)$,

$$(1)\ Time_{\sigma_1}(e) = Time_{\sigma_2}(h(e));$$

$$(2)\ l_1|_{\sigma_1}(e) = l_2|_{\sigma_2}(h(e));$$

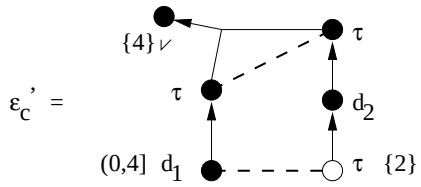
$$(3)\ e \rightarrow_1 |_{\sigma_1} e' \text{ iff } h(e) \rightarrow_2 |_{\sigma_2} h(e').$$

Clearly, when $\sigma_1 \approx \sigma_2$, except for the ev en tnames the accompanying time instants, action labels, and observable causality relations of even ts remain the same.

By $OC(\mathcal{E}_1) \approx OC(\mathcal{E}_2)$ we denote the fact that any observational configuration of $\mathcal{E}_1$ has an isomorphic correspondence in $\mathcal{E}_2$, and vice v ersa.

Definition 2.2.5 (*Observational p omset trac eequivalenc*e) $\mathcal{E}_1$ and $\mathcal{E}_2$ are observational pomset trace equivalent, denoted $\mathcal{E}_1 \cong_p \mathcal{E}_2$, iff $OC(\mathcal{E}_1) \approx OC(\mathcal{E}_2)$.

Example 2.2.6 $\mathcal{E}_c \cong_p \mathcal{E}'_c$, where $\mathcal{E}'_c$ is the tes below, and $\mathcal{E}_c$ is the tes of Example 2.2.2.



2.2.2 A denotational semantics

We first describe the operators on TES corresponding to those in the timed process algebra. Here we only present the definitions of *p ar allelc omposition* $\|_A$, *timeout* $\triangleright_t$ and *abstraction* $\backslash A$ in detail. F or the definitions of*action prefix* $a_T.$, *relabelling* $[\lambda]$, *sequential composition* ; and *choice* $+$, we refer the reader to [12,13,14].

P arallel composition: $\mathcal{E}_1 \|_A \mathcal{E}_2 = (E, \sharp, \mapsto, l, \mathcal{D}, \mathcal{R}, \mathcal{U})$, where
$E = (E_1^f \times \{*\}) \cup (\{*\} \times E_2^f) \cup \{(e_1, e_2) \in E_1^s \times E_2^s \mid l_1(e_1) = l_2(e_2)\}$
where $E_i^s = \{e \in E_i \mid l_i(e) \in A \cup \{\surd\}\}$ and $E_i^f = E_i \setminus E_i^s$ $(i = 1, 2)$,
$(e_1, e_2)\sharp(e_1', e_2')$ iff $(e_1 \sharp_1 e_1') \vee (e_2 \sharp_2 e_2') \vee$
$(e_1 = e_1' \neq * \wedge e_2 \neq e_2') \vee (e_2 = e_2' \neq * \wedge e_1 \neq e_1')$,
$X \mapsto (e_1, e_2)$ iff $(\exists X_1 : X_1 \mapsto_1 e_1 \wedge X = \{(e, e') \in E \mid e \in X_1\}) \vee$

$(\exists X_2 : X_2 \mapsto_1 e_2 \wedge X = \{(e, e') \in E \mid e' \in X_2\})$,
$l(e_1, e_2) =$ if $e_1 = *$ then $l_2(e_2)$ else $l_1(e_1)$,
$\mathcal{D}(e_1, e_2) = \mathcal{D}_1(e_1) \cap \mathcal{D}_2(e_2)$ with $\mathcal{D}_i(*) = [0, \infty)$ $(i = 1, 2)$,
$\mathcal{R}(X, (e_1, e_2)) = \cap_{X_1 \in Pr_1} \mathcal{R}_1(X_1, e_1) \cap \cap_{X_2 \in Pr_2} \mathcal{R}_2(X_2, e_2)$ with
$\mathrm{Pr}_1 = \{X_1 \subseteq E_1 \mid X_1 \mapsto_1 e_1 \wedge X = \{(e, e') \in E \mid e \in X_1\}\}$ and
$Pr_2 = \{X_2 \subseteq E_2 \mid X_2 \mapsto_2 e_2 \wedge X = \{(e, e') \in E \mid e' \in X_2\}\}$,
$(e_1, e_2) \in \mathcal{U}$ iff $e_1 \in \mathcal{U}_1 \vee e_2 \in \mathcal{U}_2$ with $* \notin \mathcal{U}_i$ $(i = 1, 2)$.

Timeout: $\mathcal{E}_1 \rhd_t \mathcal{E}_2 = \mathcal{E}_1 + \hat{\tau}_{\{t\}}.\mathcal{E}_2$, where $\hat{\tau}$ is the urgent variant of τ. That means the event corresponding to it is labelled with action τ and it is declared to be urgent.

Since actions are no longer instantaneous, the abstraction operator $\backslash A$ has to be modified. In fact, both relabelling and abstraction can be viewed as certain action refinements for the system. Our idea to abstract an observable action a is that we insert two τ-events, one at the time point at which a starts and the other at the time point at which a finishes.

Abstraction: $\mathcal{E}_1 \backslash A = f(\mathcal{E}_1)$, the refined tes of $\mathcal{E}_1$, where $\Omega_0 = \Omega \backslash A$, and for $a \in A$, $f(a) = (e_\surd, \emptyset, \emptyset, \{(e_\surd, \surd)\}, \{(e_\surd, \{k(a)\})\}, \emptyset, \emptyset)$, namely tes $\bullet\surd\{k(a)\}$. We suggest the reader come back to this definition after reading Section 3.

The definition of tes's is independent of the durations of actions. As it has been shown in e.g. [12], all the operators mentioned are well defined, i.e. $\circ\mathcal{E}_1 (\circ \in \{a_T., [\lambda]\})$ and $\mathcal{E}_1 \circ \mathcal{E}_2 (\circ \in \{;, +, \|_A, \rhd_t\})$ are all tes's. $\mathcal{E}_1 \backslash A$ is also a tes by Theorem 3.2.5.

We are interested in expressions that are *closed*. That is, any process variable x that occurs in such an expression is in the scope of a μx operator. Let s be the cpo-based denotational semantics proposed in [12]. Basically, the semantic model of a closed expression is a tes, which is derived as follows:

$$s(0) = (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset);$$
$$s(1_T) = (\{e_\surd\}, \emptyset, \emptyset, \{(e_\surd, \surd)\}, \{(e_\surd, T)\}, \emptyset, \emptyset);$$
$$s(\circ P) = \circ s(P) \text{ for } \circ \in \{a_T., \backslash A, [\lambda]\};$$
$$s(P_1 \circ P_2) = s(P_1) \circ s(P_2) \text{ for } \circ \in \{;, +, \|_\surd, \rhd_t\}; \text{ and}$$

the semantic model of recursion is defined as the least upper bound of a set of tes's with a complete partial order (cpo) [9,12]:

$$s(\mu x.P) = (\cup_i E_i, \cup_i \sharp_i, \mapsto, \cup_i l_i, \cup_i \mathcal{D}_i, \mathcal{R}, \cup_i \mathcal{U}_i), \text{ where}$$

$\bot = (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset), v_P(v_P^{i-1}(\bot)) = v_P^i(\bot) = (E_i, \sharp_i, \mapsto_i, l_i, \mathcal{D}_i, \mathcal{R}_i, \mathcal{U}_i)$,
$\mapsto = \{(\cup_k X_k, e) \mid \forall k : (e \in E_k \Rightarrow X_k \mapsto_k e) \wedge (X_{k+1} \cap E_k = X_k)\}$,

$$\mathcal{R} = \{((\cup_k X_k, e), T_k) \mid \forall k : (e \in E_k \Rightarrow \mathcal{R}_k(X_k, e) = T_k) \wedge (X_{k+1} \cap E_k = X_k)\},$$

and v_P is defined as the function that substitutes a tes for each occurrence of x in P, interpreting all operators in P as operators on tes's.

Remark that $s(1_T)$ is a timed modification of $s(1)$ defined in e.g. [12]. In the follo wing,when the context depends on $s(P)$ we always suppose that expression P is closed.

Example 2.2.7 Suppose that P and P_c are the expressions of Example 2.1.2. Let $\mathcal{E}$ and $\mathcal{E}'_c$ be the tes's of Example 2.2.2 and Example 2.2.6, respectively . Then $s(P) = \mathcal{E}$, and

$s(P_c) = \mathcal{E}'_c$.

Hereafter, we alwa ys denote $s(P)$ by $\mathcal{E}$. An observational configuration of P means an observational configuration of $\mathcal{E}$, and $OC(P) = OC(\mathcal{E})$. Furthermore, $P_1 \cong_p P_2$ iff $s(P_1) \cong_p s(P_2)$.

3 Refinement of actions

Now, we define action refinement in our real-time process algebra. Proofs of the results in this and in the next section are sketched in the appendix.

3.1 Definition

Firstly, we hav e to sort out some "bound" actions. For a given expression P, $Sort(P)$ is defined as the smallest action set containing A and $dom(\lambda)$ if the abstraction operator $\backslash A$ and respectively the relabelling operator $[\lambda]$ occurs in P. All actions in $Sort(P)$ are not allo wed to be refined, since they may lead to a confusion of the communication levels like in the untimed case [8].

Let Ω_0 be a subset of Ω representing the set of actions that need not or cannot be refined such that $Sort(P) \cup \{\tau, \surd\} \subseteq \Omega_0$, and let $g : \Omega \setminus \Omega_0 \to TPA$ a function. For conv enience, we also use $A_{g(a)}$ to denote the action singleton $\{a\}$ when $a \in \Theta \cap \Omega_0$.

Definition 3.1.1 (*Refinement function for expression P*)

g is a refinement function for 0, 1_T and x;

g is a refinement function for $a_T.P_1$ iff g is a refinement function for P_1;

g is a refinement function for $P_1 \circ P_2$ iff g is a refinement function for P_1 and P_2, where $\circ \in \{;, +, \triangleright_t\}$;

g is a refinement function for $P_1 \parallel_A P_2$ iff g is a refinement function for P_1 and P_2, and for any two distinct $a \in A$ and $b \in \Theta$, $A_{g(a)} \cap A_{g(b)} = \emptyset$;

g is a refinement function for $\circ P_1$ iff g is a refinement function for P_1, and for any $a \in \Omega \setminus \Omega_0$, $A_{g(a)} \cap Sort(P) = \emptyset$, where $\circ \in \{\backslash A, [\lambda]\}$;

g is a refinement function for $\mu x.P_1$ iff g is a refinement function for P_1.

We forbid the actions in $Sort(P)$ to occur in $g(a)$, and we pose one more constraint in the definition for parallel composition. Usually, like the untimed case such constraints cannot be avoided in refinement on syntactic level. Otherwise they may lead to a confusion of the communication levels [8].

Example 3.1.2 Suppose that P and P_c are the expressions of Example 2.1.2, and $\Omega_0 = \Omega \setminus \{c\}$. Let $g : c \mapsto P_c$. Then g is a refinement function for P.

From now on let g represent a refinement function for expression P. We proceed to consider how g can be applied to P to obtain a refined expression. Our basic idea is that the refined expression of $a_T.P_1$ where $a \notin \Omega_0$ is defined as the sequential composition of $\tau_{T-k(a)}.g(a)$ and the refined expression of P_1.

An intuitive explanation of the prefix $\tau_{T-k(a)}$ can be given as follows: Every process has a start point, which is supposed to be performing the internal silent action. The prefix τ can be viewed as the start point of process $g(a)$. Moreover, in a run of process $a_T.P_1$, if action a finishes at time point $t \in T$, then since its execution costs $k(a)$ time units, it should start at time point $t - k(a)$. So the prefix τ is labelled with the time attachment $T - k(a)$.

Definition 3.1.3 (*Syntactic refinement of expression* P) The refinement $g(P)$ of expression P is defined as follows:

$$g(0) = 0, g(1_T) = 1_T, g(x) = x,$$

$$g(a_T.P_1) = \begin{cases} a_T.g(P_1) \text{ if } a \in \Omega_0, \\ (\tau_{T-k(a)}.g(a)); g(P_1) \text{ otherwise,} \end{cases}$$

$$g(\circ P_1) = \circ g(P_1), \text{ where } \circ \in \{\backslash A, [\lambda]\},$$

$$g(P_1 \circ P_2) = g(P_1) \circ g(P_2), \text{ where } \circ \in \{;, +, \triangleright_t\},$$

$$g(P_1 \parallel_A P_2) = g(P_1) \parallel_{g(A)} g(P_2), \text{ where } g(A) = \cup_{a \in A} A_{g(a)},$$

$$g(\mu x.P_1) = \mu x.g(P_1).$$

Theorem 3.1.4 Suppose that $P \in TPA$ and g is a refinement function for P. Then $g(P) \in TPA$.

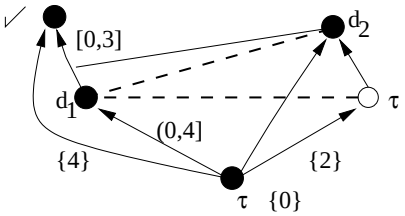
Obviously, $g(P)$ is closed if P is closed.

Example 3.1.5 Consider P and g of Example 3.1.2.
$g(P) = (a_{(0,3]}.(\tau_{[0,1]}.g(c)); 0) \parallel_{\{d_1,d_2\}} ((\tau_{[0,4]}.g(c)); 0 + b_{(0,3]}.0)$
$= (a_{(0,3]}.(\tau_{[0,1]}.(((d_{1(0,4]}.1 \triangleright_2 d_2.1); 1) \parallel_\emptyset 1_{\{4\}})); 0) \parallel_{\{d_1,d_2\}} (((\tau_{[0,4]}.(((d_{1(0,4]}.1 \triangleright_2 d_2.1); 1) \parallel_\emptyset 1_{\{4\}})); 0) + b_{(0,3]}.0)$.

3.2 Semantic correspondence

We call $\tau_{\{0\}}.\mathcal{E}$, denoted $r(\mathcal{E})$, the *rooted tes* associated with $\mathcal{E}$. In $r(\mathcal{E})$, the new event that corresponds to the prefix τ is called the *start-event* of $\mathcal{E}$ (or $r(\mathcal{E})$) and is denoted by $o_{r(\mathcal{E})}$. This new event resembles the s- and $*$-events used in [5,10,19,21] and can also be understood as the start point of the system, which is executing the internal silent action at time instant zero.

Example 3.2.1 Let $\mathcal{E}_c$ be the tes given in Example 2.2.2. Then the tes $r(\mathcal{E}_c)$ is depicted below.



The refinement of an (observable) action a, say $\mathcal{E}_a$, should be a tes. Since action $\surd$ only represents successful termination of a process, a system run, if it is not a deadlock, should contain exactly one $\surd$-event. So we require that $\mathcal{E}_a$ is well-labelled. Furthermore,if action a is observable, we require that its refinement $\mathcal{E}_a$ is "observable" as well. That is, $\mathcal{E}_a$ should be non-forgetful. Our requirement on time is that, the "duration" of $\mathcal{E}_a$ should be the duration of action a, i.e. each successful termination run of $\mathcal{E}_a$ have to last exactly $k(a)$ time units. In other words, if it starts at time point 0, then it should successfully terminate exactly at time point $k(a)$.

Definition 3.2.2 (*Semantic refinement function*) $f : \Omega \setminus \Omega_0 \to TES$ is called a refinement function if for any action $a \in \Omega \setminus \Omega_0$, the following conditions are met:

(1) $f(a)$ is well-labelled and non-forgetful;

(2) $maxt(f(a)) = \{k(a)\}$.

We call $f(a)$ the *refinement of action* a (with respect to f). In practice, it is reasonable to require that $f(a)$ is finite. If so, the conditions in this definition can be checked and easily enforced. This definition of action refinement is rather strict. Processes that are slower or quicker than the duration of action a are not allowed to be valid refinements of a. We do this partly for technical reasons, and partly to emphasize that slow-down is certainly not desirable and speed-up is also not always desirable [2].

Example 3.2.3 Suppose $\Omega \setminus \Omega_0 = \{c\}$, and $\mathcal{E}_c$ is the tes of Example 2.2.2.

Then $f(c) = \mathcal{E}_c$ is a refinement of action c.

Let f represent a semantic refinement function. Now we define how f is used to refine a tes. For simplicity, we use $rfl(e)$ and $rf(a)$ to abbreviate $r(f(l(e)))$ and $r(f(a))$, respectively.

Definition 3.2.4 (*Refinement of a tes*) The refinement of $\mathcal{E}$ is defined as $f(\mathcal{E}) = (E_f, \sharp_f, \mapsto_f, l_f, \mathcal{D}_f, \mathcal{R}_f, \mathcal{U}_f)$, where

- $E_f = \{(e, e') \mid (e \in E) \wedge (l(e) \notin \Omega_0) \wedge (e' \in E_{rfl(e)})\} \cup$
  $\{(e, e) \mid (e \in E) \wedge (l(e) \in \Omega_0)\}$,

- $\forall (e_1, e_2) \in E_f \forall (e_1', e_2') \in E_f,\ (e_1, e_2) \sharp_f (e_1', e_2') \Leftrightarrow$
  $(e_1 = e_1') \Rightarrow (e_2 \sharp_{rfl(e_1)} e_2') \vee (e_2, e_2' \in exit(rfl(e_1)) \wedge e_2 \neq e_2')$,
  $(e_1 \neq e_1') \Rightarrow$
  $(e_2 = e_1) \wedge (e_2' = e_1') \Rightarrow (e_1 \sharp e_1')$,
  $(e_2 \neq e_1) \wedge (e_2' = e_1') \Rightarrow (e_1 \sharp e_1') \wedge (e_2 \in \{o_{rfl(e_1)}\} \cup exit(rfl(e_1)))$,
  $(e_2 = e_1) \wedge (e_2' \neq e_1') \Rightarrow (e_1 \sharp e_1') \wedge (e_2' \in \{o_{rfl(e_1')}\} \cup exit(rfl(e_1')))$,
  $(e_2 \neq e_1) \wedge (e_2' \neq e_1') \Rightarrow (e_1 \sharp e_1') \wedge (e_2 \in \{o_{rfl(e_1)}\} \cup exit(rfl(e_1))) \wedge$
  $(e_2' \in \{o_{rfl(e_1')}\} \cup exit(rfl(e_1')))$,

- $\forall X \subseteq E_f\ \forall (e_1, e_2) \in E_f,\ X \mapsto_f (e_1, e_2) \Leftrightarrow$
  $(e_2 \neq e_1) \wedge (e_2 \in E_{rfl(e_1)} \setminus \{o_{rfl(e_1)}\}) \Rightarrow$
  $(\pi_1(X) = \{e_1\}) \wedge (\pi_2(X) \mapsto_{rfl(e_1)} e_2)$,
  $((e_2 \neq e_1) \wedge (e_2 \in \{o_{rfl(e_1)}\} \cup exit(rfl(e_1)))) \vee (e_2 = e_1) \Rightarrow$
  $(\pi_1(X) \mapsto e_1) \wedge (\pi_2(X) = \cup_{e \in \pi_1(X), l(e) \notin \Omega_0} exit(rfl(e)) \cup (\cup_{e \in \pi_1(X), l(e) \in \Omega_0} \{e\}))$,

- $\forall (e_1, e_2) \in E_f,\ (e_2 \neq e_1) \Rightarrow$
  $(e_2 \notin exit(rfl(e_1))) \Rightarrow (l_f(e_1, e_2) = l_{rfl(e_1)}(e_2))$,
  $(e_2 \in exit(rfl(e_1))) \Rightarrow (l_f(e_1, e_2) = \tau)$,
  $(e_2 = e_1) \Rightarrow (l_f(e_1, e_2) = l(e_1))$,

- $\forall (e_1, e_2) \in E_f,\ (e_2 = e_1) \Rightarrow (\mathcal{D}_f(e_1, e_2) = \mathcal{D}(e_1))$,
  $(e_2 \neq e_1) \Rightarrow$
  $(e_2 = o_{rfl(e_1)}) \Rightarrow (\mathcal{D}_f(e_1, e_2) = (\mathcal{D}(e_1) - k(l(e_1))))$,
  $(e_2 \neq o_{rfl(e_1)}) \Rightarrow (\mathcal{D}_f(e_1, e_2) = [0, \infty))$,

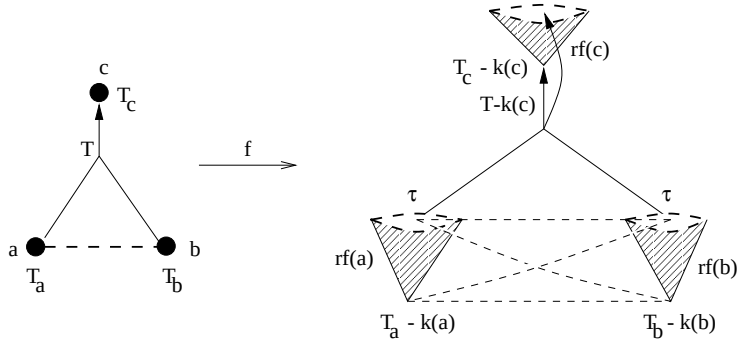
- $\forall X \subseteq E_f\ \forall (e_1, e_2) \in E_f,\ (X \mapsto_f (e_1, e_2)) \Rightarrow$
  $(e_2 \neq e_1) \wedge (e_2 \in E_{rfl(e_1)} \setminus \{o_{rfl(e_1)}\}) \wedge \pi_1(X) = \{e_1\}) \Rightarrow$
  $(\mathcal{R}_f(X, (e_1, e_2)) = \mathcal{R}_{rfl(e_1)}(\pi_2(X), e_2))$,
  $(e_2 \neq e_1) \wedge (e_2 = o_{rfl(e_1)}) \Rightarrow$
  $(\mathcal{R}_f(X, (e_1, e_2)) = (\mathcal{R}(\pi_1(X), e_1) - k(l(e_1))))$,

$(e_2 \neq e_1) \wedge (e_2 \in exit(rfl(e_1))) \wedge (\pi_1(X) \neq \{e_1\}) \Rightarrow$
$(\mathcal{R}_f(X,(e_1,e_2)) = [0,\infty))$,
$(e_2 = e_1) \Rightarrow (\mathcal{R}_f(X,(e_1,e_2)) = \mathcal{R}(\pi_1(X),e_1))$,

- $\forall (e,e') \in E_f$, $(e,e') \in \mathcal{U}_f \Leftrightarrow (e \in \mathcal{U}) \vee (e' \neq e \Rightarrow e' \in \mathcal{U}_{rfl(e)})$.

Here $\pi_1(X) = \{e \mid (e,e') \in X\}$ and $\pi_2(X) = \{e' \mid (e,e') \in X\}$.

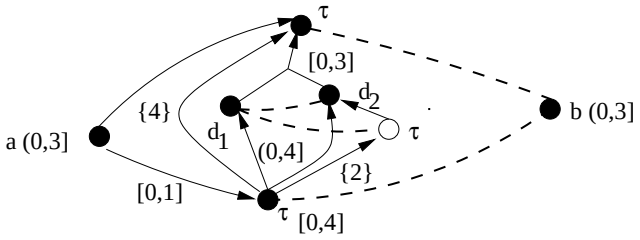
The following figure demonstrates how $f(\mathcal{E})$ is obtained.



The definition of the set of events, the conflict relations, the bundle relations as well as the action labels of events is easily understood. Here we say something about the timing information and urgent events. For convenience, we assume without loss of generality $k(a) \leq T_a$, $k(b) \leq T_b$, $k(c) \leq T_c$ and $k(c) \leq T$. Since $rf(c)$ lasts exactly $k(c)$ time units and c finishes at some time point, say t, in T_c, in the refined tes, the start-event of $rf(c)$ should finish at time point $t - k(c)$. So the event delay of start-event of $rf(c)$ should be $T_c - k(c)$. In a run of the original tes, if the delay from one event in the bundle-set $\{a,b\}$ to c is t time units, then $t \in T$, and in the corresponding run of the refined tes the delay from one event in the newly derived bundle-set $exit(rf(a)) \cup exit(rf(b))$ to the start-event of $rf(c)$ should be $t - k(c)$ time units. Therefore, the bundle delay from $exit(rf(a)) \cup exit(rf(b))$ to the start-event of $rf(c)$ should be $T - k(c)$. The event delays of the start-events of $rf(a)$ and $rf(b)$ can be explained similarly. Urgent events of the refined tes are the conjunction of urgent events of the original tes and urgent events of the tes's substituted for actions. To guarantee constraint (2) in the definition of tes's, we have to introduce an arrow from the bundle-set $exit(rf(a)) \cup exit(rf(b))$ to each event of $exit(rf(c))$ as well as the dotted lines between the events of $exit(rf(a))$ $[exit(rf(b))]$ and the start-event of $rf(b)$ [resp. $rf(a)$].

Theorem 3.2.5 Suppose that $\mathcal{E} \in TES$ and f is a refinement function. Then $f(\mathcal{E}) \in TES$.

Example 3.2.6 Let $\mathcal{E}$ be the tes of Example 2.2.2 and $f(c)$ be the refinement of action c defined in Example 3.2.3. Then $f(\mathcal{E})$ is the tes presented below.



4 Properties

Suppose $g : \Omega \setminus \Omega_0 \to TPA$ is a syntactic refinement function for a giv enexpression P, and $f : a \mapsto s(g(a))$ $(a \in \Omega \setminus \Omega_0)$ is a semantic refinement function. Remark that $s(g(a))$ is naturally well-labelled, and when no real multi-wa y synchronization appears in $g(a)$, a syntactic characterization of $s(g(a))$ with property $maxt(s(g(a))) = \{k(a)\}$ is possible if $s(g(a))$ satisfies the so-called finite v ariability or non-Zeno property [3,20].

Theorem 4.1.1 (*Coincidence result*) Suppose that $P \in TPA$, $g : \Omega \setminus \Omega_0 \to TPA$ is a syntactic refinement function for P, and $f : \Omega \setminus \Omega_0 \to TES, f(a) = s(g(a))$ $(a \in \Omega \setminus \Omega_0)$ is a semantic refinement function. Then $f(s(P)) \cong_p s(g(P))$, where s is the cpo-based semantics mentioned in Section 2.2.

This theorem demonstrates that our syntactic and semantic refinement operations coincide up to observational pomset trace equivalence with respect to the cpo-based denotational semantics. Because of this result, under observational pomset trace equivalence we do not need to distinguish f from g. Note that Theorem 4.1.1 may not hold if we do not abstract a wa y τ-ev en ts. The main reason is that we can by no means require that some τ-actions can be executed simultaneously. The constraint that actions in $Sort(P)$ are not allo wed to be refined and the constraint for refinement of parallel composition are also necessary.

Now, let σ be an observational configuration of P, $e \in E(\sigma)$, $l(e) \notin \Omega_0$, and σ_e be an observational configuration of the expression, say P_e, which is used to substitute for that action labelled to ev en te. Moreover, we suppose that P_e satisfies the condition that σ_e successfully terminates if event e is not maximal in σ. Since the event labelled with action $\surd$ is relabelled with action τ b y the refinement procedure, we remov e such event from σ_e. Let

$$\sigma_g = \{((e, e_j), t_j) \mid e \in E(\sigma) \text{ and } \text{if } l(e) \in \Omega_0 \text{ then } e_j = e,\ t_j = Time_\sigma(e)$$
$$\text{otherwise } e_j \in E(\sigma_e),\ t_j = Time_{\sigma_e}(e_j) + (Time_\sigma(e) - k(l(e)))\}.$$

We call σ_g a *refinement of* σ. It is derived by replacing each timed event (e, t) in observational configuration σ with $l(e) \notin \Omega_0$ by an observational configuration σ_e of P_e, where the accompanying time instants of events are adjusted according to the accompanying time instant t of event e. Since the newly introduced start-events are labelled with action τ, they do not appear in the refinement of the observational configurations anymore.

Theorem 4.1.2 (*Safety*) Let g be a refinement function for $P \in TPA$. Then $OC(g(P)) = \{\sigma_g \mid \sigma_g \text{ is a refinement of } \sigma \in OC(P)\}$.

Remark that all the constraints in the definition of semantic refinement functions are needed for the correctness of this theorem. It is straightforward to see that the observable causality relations in each σ_e are respected in the corresponding refinement of σ. The observable causality relations in σ are respected in the meaning that if e observationally causes e' in σ, then some maximal events of σ_e cause the minimal events of $\sigma_{e'}$. The observable behaviour of refined expression can be thus inferred compositionally from the observable behaviour of the original one and from the observable behaviour of those substituted for the actions. Our refinement of actions is safe in this sense.

Theorem 4.1.3 (*Congruence result*) Let g_1 and g_2 be refinement functions for two expressions P_1 and P_2, respectively. If $P_1 \cong_p P_2$, and for any $a \in \Omega \setminus \Omega_0$, $g_1(a) \cong_p g_2(a)$, then $g_1(P_1) \cong_p g_2(P_2)$.

This theorem indicates that observational pomset trace equivalence is a congruence under our refinement. Remark that the conclusion also holds for the standard process operators. See Appendix for the detailed description.

Theorems 4.1.1, when no synchronization actions are allowed to be refined, and Theorem 4.1.3 also hold for observational history preserving bisimulation equivalence.

Example 4.1.4 For the refinement $g(P)$ of expression P described in Example 3.1.5, $s(g(P))$ is the tes depicted in the following,which is pomset trace equivalent to tes $f(\mathcal{E})$ of Example 3.2.6.

5 Concluding remarks

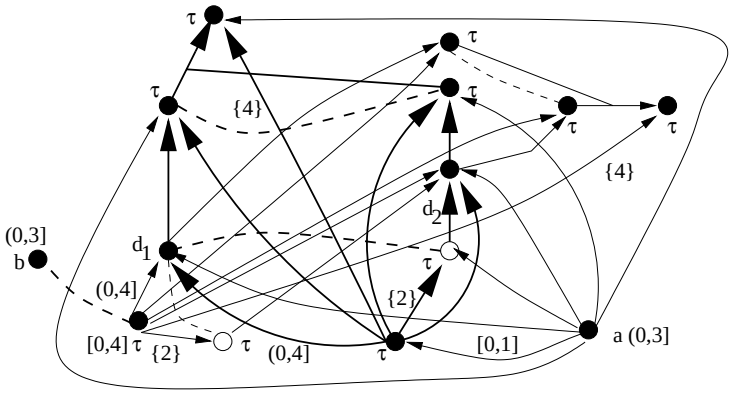
In this paper, we developed the concepts of syntactic and semantic action refinements in a real-time process algebra with a true concurrency model. We

showed that they are safe and conform to each other up to e.g. observational pomset trace equivalence with respect to a cpo-based denotational semantics. Also, we demonstrated that such equivalence relations are congruences under our refinement operations.

The refinement operator defined by Glabbeek and Goltz [7] for various untimed event structures does not work for our tes's. On the one hand, it is difficult to deal with timing information, especially those attached to events. On the other hand, the property of urgent events in the definition of tes's is violated. A counter-example can be constructed easily by considering the tes $\mathcal{E}$ and the refinement function f presented in Example 3.1.2.

Murphy and Pitt [19] defined a timed variant of prime event structures with a notion of refinement. From the functional point of view, bundle event structures are more expressive than prime event structures. From the timing point of view, [19] is a special case of tes's when each of the events and bundles is attached with only one time instant. Consequently, our systems are more flexible and realistic.

In [17] the authors developed a technique of action refinement in real-time concurrent systems without multi-way synchronization. This technique cannot tackle, for instance, the case of Example 3.1.2.



Appendix

We sketch proofs of our theorems in this part. Theorems 3.1.4 and 3.2.5 follow directly from the definitions of the expressions, the tes's and the corresponding refinement operations. We hereby focus on the remaining statements.

Lemma 1 Suppose $\mathcal{E}_1 \cong_p \mathcal{E}_2$, $\mathcal{E}_3 \cong_p \mathcal{E}_4$. Then

$\circ\mathcal{E}_1 \cong_p \circ\mathcal{E}_2$ $(\circ \in \{a_T., \backslash A, [\lambda]\})$;

$\mathcal{E}_1 \circ \mathcal{E}_3 \cong_p \mathcal{E}_2 \circ \mathcal{E}_4$ $(\circ \in \{;, \|_A, \rhd_t\})$.

The conclusion does not hold for the choice operator. This is resulted from

neglecting the applications of the initial urgent events. It can be avoided when such urgent events as well as some of their timing constraints are required to remain unchanged.

Let $\mathcal{E}_1 \cong_p \mathcal{E}_2$, and σ_1 and σ_2 be the observational configurations in the definition of observational pomset trace equivalence. Assume that $D(\sigma_i)$ $(i = 1, 2)$ consists of the configurations of $\mathcal{E}_i$ from which σ_i is derived. For $\sigma \in D(\sigma_i)$, let

$$iU(\sigma) = E(\sigma) \cap init(\mathcal{E}_i) \cap \mathcal{U}_i \text{ and } minT(iU(\sigma)) = min\{Time_\sigma(e) \mid e \in iU(\sigma)\}.$$

We say that $\mathcal{E}_1$ and $\mathcal{E}_2$ are *initial-urgency-invariant*, if

$$\exists \sigma' \in D(\sigma_1) : iU(\sigma') \neq \emptyset \Leftrightarrow \exists \sigma'' \in D(\sigma_2) : iU(\sigma'') \neq \emptyset, \text{ and}$$
$$minT(iU(\sigma')) = minT(iU(\sigma'')).$$

For such pairs of tes's, we have

Lemma 2 $\mathcal{E}_1 + \mathcal{E}_3 \cong_p \mathcal{E}_2 + \mathcal{E}_4$, if $\mathcal{E}_i$ and $\mathcal{E}_{i+1}$ $(i \in \{1, 3\})$ are initial-urgency-invariant.

The above two lemmas indicate that in general observational pomset trace equivalence is a congruence under the standard operators on tes's.

Lemma 3 below shows that introducing the start-event into a tes does not have influence on the observational behaviours and on the timing information of the termination of the system runs. Additionally, it does not alter the properties of being non-forgetful and well-labelled.

Lemma 3 (1) $\mathcal{E}$ is non-forgetful iff $r(\mathcal{E})$ is non-forgetful;
(2) $\mathcal{E}$ is well-labelled iff $r(\mathcal{E})$ is well-labelled;
(3) $OC(\mathcal{E}) = OC(r(\mathcal{E}))$ and $\mathcal{E} \cong_p r(\mathcal{E})$;
(4) $maxt(\mathcal{E}) = maxt(r(\mathcal{E}))$.

Now suppose that σ_f is a configuration of $f(\mathcal{E})$. Let

$$\pi_1(\sigma_f) = \{(e, t) \mid \text{ there exists } (e, e_j) \in E(\sigma_f) \text{ and if } l(e) \notin \Omega_0 \text{ then } t = Time_{\sigma_f}(e, o_{rfl(e)}) + k(l(e)) \text{ otherwise } t = Time_{\sigma_f}(e, e)\}.$$

For $e \in E(\pi_1(\sigma_f))$ with $l(e) \notin \Omega_0$, let

$$\pi_2(\sigma_f, e) = \{(e_j, t_j) \mid (e, e_j) \in E(\sigma_f) \text{ and } t_j = Time_{\sigma_f}(e, e_j) - Time_{\sigma_f}(e, o_{rfl(e)})\}.$$

$\pi_1(\sigma_f)$ is the projection of σ_f on $\mathcal{E}$, and $\pi_2(\sigma_f, e)$ the projection of σ_f on $rfl(e)$, where the accompanying time instants of events are adjusted according to the

time instant at which the start-event of each $f(l(e))$ occurs in σ_f. From the definition of configurations together with Lemma 3 (4), we have the following

Lemma 4 (1) $\pi_1(\sigma_f)$ is a configuration of $\mathcal{E}$; (2) $\pi_2(\sigma_f, e)$ is a configuration of $rfl(e)$ and it successfully terminates if e is not maximal in $E(\pi_1(\sigma_f))$.

Lemma 5 $OC(f(a_T.s(P_1))) = OC(a_T.f(s(P_1)))$, if $a \in \Omega_0$;

$OC(f(a_T.s(P_1))) = OC((\tau_{T-k(a)}.f(a)); f(s(P_1)))$, if $a \notin \Omega_0$;

$OC(f(\circ s(P_1))) = OC(\circ f(s(P_1))), \circ \in \{\backslash A, [\lambda]\}$;

$OC(f(s(P_1) \circ s(P_2))) = OC(f(s(P_1)) \circ f(s(P_1))), \circ \in \{;, +, \triangleright_t\}$.

Lemma 5 follows from Lemma 4and our requirement that the actions involved with abstraction and relabelling operators are not allowed to be refined.

In Lemma 5, if we denote the tes appearing in the left-hand side of an equality by $\mathcal{E}_f$ and that in the corresponding right-hand side by $\mathcal{E}_g$, then for any observational configuration σ of $\mathcal{E}_f$, $\rightarrow_f |_\sigma = \rightarrow_g |_\sigma$ and $l_f|_\sigma = l_g|_\sigma$. Thus, trivially $OC(\mathcal{E}_f) \approx OC(\mathcal{E}_g)$, and $\mathcal{E}_f \cong_p \mathcal{E}_g$.

Similarly, using Lemmas 4 and the condition that $A_{g(a)} \cap A_{g(b)} = \emptyset$ for two distinct $a \in A$ and $b \in \Theta$, we have the following lemma for parallel composition. Remark that it does not hold if τ-events are not abstracted away.

Lemma 6 $f(s(P_1) \parallel_A s(P_2)) \cong_p f(s(P_1)) \parallel_{g(A)} f(s(P_2))$.

Theorem 4.1.1 follows from Lemmas 1, 2, 5 and 6. Indeed, $f(s(P))$ and $s(g(P))$ are initial-urgency-invariant. The conclusion for recursion follows from the fact that $f(v_P^k(\perp)) \cong_p v_{g(P)}^k(\perp)$ for $k \geq 0$. This fact can be proved by means of the conclusions for the other operators. Theorem 4.1.2 follows from Lemma 3 and Lemma 4. Finally, Theorem 4.1.3 is concluded by Lemma 4 and Theorem 4.1.2.

References

[1] Aceto, L, *Action Refinement in Process Algebra*, Cambridge Univ. Press, 1992.

[2] Aceto, L, Murphy, D., Timing and Causality in Process Algebra, *Acta Informatica*, 33: 317 – 350, 1996.

[3] Alur, R., Dill, D. L., A Theory of Timed Automata, *Th. Comp. Sci.*, 126: 183 – 235, 1994.

[4] Bolognesi, T., Brinksma, E., Introduction to the ISO Specification Language LOTOS, *Comp. Netw. & ISDN Syst.*, 14: 25 – 59, 1987.

[5] Bowman, H., Derrick, J., Extending LOTOS with Time: A True Concurrency Perspective, *Lecture Notes in Computer Science*, 1231: 383 – 399, 1997.

[6] Bowman, H., Katoen, J-P., A True Concurrency Semantics for ET-LOTOS, *Proceedings Int. Conference on Applications of Concurrency to System Design*, 228 – 239, 1998.

[7] van Glabbeek, R., Goltz, U., Refinement of Actions and Equivalence Notions for Concurrent Systems, *Acta Informatica*, 37: 229 – 327, 2001.

[8] Gorrieri, R., Rensink, A., Action Refinement, *Handbook of Process Algebra*, Elsevier Science, 1047 – 1147, 2001.

[9] Fecher, H., Majster-Cederbaum, M. & Wu, J., Bundle Event Structures: A Revised Cpo Approach, *Information Processing Letters*, 83: 7 – 12, 2002.

[10] Fecher, H., Majster-Cederbaum, M. & Wu, J., Action Refinement for Probabilistic Processes with True Concurrency Models, *PAPM-Probmiv'02*, LNCS 2399: 77 – 94, 2002.

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

[12] Katoen, J-P., *Quantitative and Qualitative Extensions of Event Structures*, PhD Thesis, University of Twente, 1996.

[13] Katoen, J-P., Baier, C. & Latella, D., Metric Semantics for True Concurrent Real Time, *Th. Comp. Sci.*, 254(1-2): 501 – 542, 2001.

[14] Katoen, J-P., Langerak, R., Brinksma, E., Latella, D. & Bolognesi, T., A Consistent Causality Based View on a Timed Process Algebra Including Urgent Interactions, *Formal Meth. in Sys. Design*, 12: 189 – 216, 1998.

[15] Langerak, R., *Transformations and Semantics for LOTOS*, PhD Thesis, University of Twente, 1992.

[16] Majster-Cederbaum, M., Salger, F., Correctness by Construction: Towards Verification in Hierarchical System Development, *Lecture Notes in Computer Science*, 1885: 163 – 180, 2000.

[17] Majster-Cederbaum, M., Wu, J., Action Refinement for True Concurrent Real Time, *Proc. ICECCS 2001*, IEEE Computer Society Press, 58 – 68, 2001.

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

[19] Murphy, D., Pitt, D., Real-Timed Concurrent Refineable Behaviours, *Lecture Notes in Computer Science*, 571: 529 – 545, 1992.

[20] Nicollin, X., Sifakis, J., An Overview and Synthesis on Timed Process Algebra, in: *Real-Time:Theory in Practice*, Lecture Notes in Computer Science, 660: 526 – 548, 1992.

[21] Žic, J., Time-Constrained Buffer Specifications in CSP+T and Timed CSP, *ACM Transactions on Programming and Systems*, 16(6): 1661 – 1674, 1994.