Relational Concurrent Refinement: Timed Refinement

John Derrick, Eerke Boiten

Relational Concurrent Refinement: timed refinement

John Derrick[1] and Eerke Boiten[2]

[1] Department of Computer Science, University of Sheffield, Sheffield, S1 4DP, UK
[2] School of Computing, University of Kent, Canterbury, Kent, CT2 7NF, UK

Abstract. Data refinement in a state-based language such as Z is defined using a relational model in terms of the behaviour of abstract programs. Downward and upward simulation conditions form a sound and jointly complete methodology to verify relational data refinements, which can be checked on an event-by-event basis rather than per trace. In models of concurrency, refinement is often defined in terms of sets of observations, which can include the events a system is prepared to accept or refuse, or depend on explicit properties of states and transitions. By embedding such concurrent semantics into a relational one, event-wise verification methods for such refinement relations can be derived. In this paper we continue our program of deriving simulation conditions for process algebraic refinement by considering how notions of time should be embedded into a relational model, and thereby deriving relational notions of timed refinement.

Keywords: Data refinement, Z, simulations, timed-refinement.

1 Introduction

The modelling and understanding of time is important in computer science. It plays an especially important role in refinement, where how time is modelled and how it is treated in a development step lead to important differences and subtleties in notions of refinement. These distinctions are more prominent in a process algebra or behavioural setting where many refinement preorders have been defined, reflecting different choices of what is taken to be observable and different choices of how time is modelled.

In a process algebra such as CSP [11] a system is defined in terms of actions (or events) which represent the *inter*actions between a system and its environment. The exact way in which the environment is allowed to interact with the system varies between different semantics. Typical semantics are set-based, associating one or more sets with each process, for example traces, refusals, divergences. Refinement is then defined in terms of set inclusions and equalities between the corresponding sets for different processes. A survey of many prominent untimed process algebraic refinement relations is given in [24,25]. The addition of time adds further complications, and there are important choices as to how time is modelled, and the assumptions one makes. These choices all affect any associated refinement relations.

In a state-based system, e.g., one specified in Z, specifications are considered to define abstract data types (ADTs), consisting of an initialisation, a collection of operations and a finalisation. A program over an ADT is a sequential composition of these elements. Refinement is defined to be the subset relation over program behaviours for all possible programs, where what is deemed visible is the input/output relation. The accepted approach to make verification of refinements tractable is through downward and upward simulations which are sound and jointly complete [4]. Although there has been some work on modelling time relationally (eg see work on duration calculus and

its integration with state-based languages [27]), there has been less work on associated refinement relations, and none on how to model the various choices that arise in a process algebraic setting.

In integrated notations, for both practical and theoretical reasons, it is important to understand how time is modelled, whether from the process algebraic or state-based angle, and how this impacts on refinement. Our ongoing research on relational concurrent refinement [6,3,7,2,8] contributes to this agenda, by explicitly recording in a relational setting the observations characterising process algebraic models. This allows the verification of concurrent refinement through the standard relational method of simulations, and to interpret relational formalisms like Z in a concurrency model.

We derived simulation rules for process algebraic refinement (such as trace, failures-divergences, readiness [6] etc), including also outputs and internal operations [3], for different models of divergence [2], as well as automaton-based refinements [8]. The current paper extends this by considering how time can be modelled in a relational context, and thus we derive simulation rules for some of the timed refinement preorders. In Section 2 we provide the basic definitions and background. In Section 3 we provide the simulation rules for a number of process algebraic preorders. In Section 4 we introduce time and timed refinements, and derive their relational simulation rules. We conclude in Section 5.

2 Background

The standard refinement theory of Z [26,5] is based on a relational model of data refinement where all operations are total, as described in [10]. However, the restriction to total relations can be dropped, see [9], and soundness and joint completeness of the same set of simulation rules in the more general case can be shown.

2.1 A partial relational model

A program (defined here as a sequence of operations) is given as a relation over a global state G, implemented using a local state State. The *initialisation* of the program takes a global state to a local state, on which the operations act, a *finalisation* translates back from local to global. In order to distinguish between relational formulations (which use Z as a meta-language) and expressions in terms of Z schemas etc., we use the convention that expressions in the relational data types are typeset in a sans serif font.

Definition 1 (Data type)
A (partial) data type *is a quadruple* $(\mathsf{State}, \mathsf{Init}, \{\mathsf{Op}_i\}_{i\in J}, \mathsf{Fin})$*. The operations* $\{\mathsf{Op}_i\}$*, indexed by* $i \in J$*, are relations on the set* State*;* Init *is a total relation from* G *to* State*;* Fin *is a total relation from* State *to* G*. If the operations are all total relations, we call it a* total *data type.* □

Insisting that Init and Fin be total merely records the facts that we can always start a program sequence and that we can always make an observation.

Definition 2 (Program)
For a data type $\mathsf{D} = (\mathsf{State}, \mathsf{Init}, \{\mathsf{Op}_i\}_{i\in J}, \mathsf{Fin})$ *a* program *is a sequence over* J*. The meaning of a program* p *over* D *is denoted by* $\mathsf{p_D}$*, and defined as follows. If* $\mathsf{p} = \langle \mathsf{p_1}, ..., \mathsf{p_n} \rangle$ *then* $\mathsf{p_D} = \mathsf{Init} \fatsemi \mathsf{Op_{p_1}} \fatsemi ... \fatsemi \mathsf{Op_{p_n}} \fatsemi \mathsf{Fin}$. □

Definition 3 (Data refinement)
For data types A *and* C, C refines A, *denoted* $\mathsf{A} \sqsubseteq_{data} \mathsf{C}$ *(dropping the subscript if the context is clear), iff for each program* p *over* J, $\mathsf{p_C} \subseteq \mathsf{p_A}$. □

Downward and *upward* simulations form a sound and jointly complete [10,4] proof method for verifying refinements. In a simulation a step-by-step comparison is made of each operation in the data types, and to do so the concrete and abstract states are related by a retrieve relation.

Definition 4 (Downward and upward simulations)
Let $\mathsf{A} = (\mathsf{AState}, \mathsf{AInit}, \{\mathsf{AOp}_i\}_{i \in J}, \mathsf{AFin})$ *and* $\mathsf{C} = (\mathsf{CState}, \mathsf{CInit}, \{\mathsf{COp}_i\}_{i \in J}, \mathsf{CFin})$. *A* downward *simulation is a relation* R *from* AState *to* CState *satisfying*

$$\mathsf{CInit} \subseteq \mathsf{AInit} \fatsemi \mathsf{R}$$
$$\mathsf{R} \fatsemi \mathsf{CFin} \subseteq \mathsf{AFin}$$
$$\forall\, i : J \bullet \mathsf{R} \fatsemi \mathsf{COp}_i \subseteq \mathsf{AOp}_i \fatsemi \mathsf{R}$$

An upward *simulation is a relation* T *from* CState *to* AState *such that*

$$\mathsf{CInit} \fatsemi \mathsf{T} \subseteq \mathsf{AInit}$$
$$\mathsf{CFin} \subseteq \mathsf{T} \fatsemi \mathsf{AFin}$$
$$\forall\, i : J \bullet \mathsf{COp}_i \fatsemi \mathsf{T} \subseteq \mathsf{T} \fatsemi \mathsf{AOp}_i$$

2.2 Refinement in Z

The definition of refinement in a specification language such as Z is usually based on the relational framework described above, using an additional intermediate step (not used in the rest of this paper) where partial relations are embedded into total relations ("totalisation", see [26,5] for details). Specifically, a Z specification can be thought of as a data type, defined as a tuple $(State, Init, \{Op_i\}_{i \in J})$. The operations Op_i are defined in terms of (the variables of) $State$ (its before-state) and $State'$ (its after-state). The initialisation is also expressed in terms of an after-state $State'$. In addition to this, operations can also consume inputs and produce outputs. As finalisation is implicit in these data types, it only has an occasional impact on specific refinement notions. If specifications have inputs and outputs, these are included in both the global and local state of the relational embedding of a Z specification. See [5] for the full details on this – in this paper we only consider data types without inputs and outputs. In concurrent refinement relations, inputs add little complication; outputs particularly complicate refusals as described in [3].

In a context where there is no input or output, the global state contains no information and is a one point domain, i.e., $\mathsf{G} == \{*\}$, and the local state is $\mathsf{State} == State$. In such a context the other components of the embedding are as given below.

Definition 5 (Basic embedding of Z data types) *The Z data type* $(State, Init, \{Op_i\}_{i \in J})$ *is interpreted relationally as* $(\mathsf{State}, \mathsf{Init}, \{\mathsf{Op}_i\}_{i \in J}, \mathsf{Fin})$ *where*

$$\mathsf{Init} == \{Init \bullet * \mapsto \theta State'\}$$
$$\mathsf{Op} == \{Op \bullet \theta State \mapsto \theta State'\}$$
$$\mathsf{Fin} == \{State \bullet \theta State \mapsto *\}$$

Given these embeddings, we can translate the relational refinement conditions of downward simulations for totalised relations into the following refinement conditions for Z ADTs.

Definition 6 (Standard downward simulation in Z)
Given Z data types $A = (AState, AInit, \{AOp_i\}_{i \in J})$ *and* $C = (CState, CInit, \{COp_i\}_{i \in J})$. *The relation* R *on* $AState \wedge CState$ *is a* downward simulation *from* A *to* C *in the non-blocking model if*

$$\forall\, CState' \bullet CInit \Rightarrow \exists\, AState' \bullet AInit \wedge R'$$
$$\forall\, i : J;\ AState;\ CState \bullet \mathrm{pre}\, AOp_i \wedge R \Rightarrow \mathrm{pre}\, COp_i$$
$$\forall\, i : J;\ AState;\ CState;\ CState' \bullet \mathrm{pre}\, AOp_i \wedge R \wedge COp_i \Rightarrow \exists\, AState' \bullet R' \wedge AOp_i$$

In the blocking model, the correctness (last) condition becomes

$$\forall\, i : J;\ AState;\ CState;\ CState' \bullet R \wedge COp_i \Rightarrow \exists\, AState' \bullet R' \wedge AOp_i$$ □

The translation of the upward simulation conditions is similar, however this time the finalisation produces a condition that the simulation is total on the concrete state.

Definition 7 (Standard upward simulation in Z)
For Z data types A *and* C, *the relation* T *on* $AState \wedge CState$ *is an* upward simulation *from* A *to* C *in the non-blocking model if*

$$\forall\, AState';\ CState' \bullet CInit \wedge T' \Rightarrow AInit$$
$$\forall\, i : J;\ CState \bullet \exists\, AState \bullet T \wedge (\mathrm{pre}\, AOp_i \Rightarrow \mathrm{pre}\, COp_i)$$
$$\forall\, i : J;\ AState';\ CState;\ CState' \bullet (COp_i \wedge T') \Rightarrow (\exists\, AState \bullet T \wedge (\mathrm{pre}\, AOp_i \Rightarrow AOp_i))$$

In the blocking model, the correctness condition becomes

$$\forall\, i : J;\ AState';\ CState;\ CState' \bullet (COp_i \wedge T') \Rightarrow \exists\, AState \bullet T \wedge AOp_i$$ □

3 Process algebraic based refinement

The semantics of a process algebra [11,14,1] is often given by associating a labelled transition system (LTS) to each term. Equivalence, and preorders, can be defined over the semantics where two terms are identified whenever no observer can notice any difference between their external behaviours. Varying how the environment *interacts* with a process leads to differing observations and therefore different preorders (i.e., refinement relations) – an overview and comprehensive treatment is provided by van Glabbeek in [24,25]. We will need the usual notation for labelled transition systems:

Definition 8 (Labelled Transition Systems (LTSs))
A labelled transition system is a tuple $L = (States, Act, T, Init)$ *where* $States$ *is a non-empty set of states,* $Init \subseteq States$ *is the set of initial states,* Act *is a set of actions, and* $T \subseteq States \times Act \times States$ *is a transition relation. The components of* L *are also accessed as* $states(L) = States$ *and* $init(L) = Init$. □

Every state in the LTS itself represents a process – namely the one representing all possible behaviour from that point onwards. Specific notation needed includes the usual notation for writing transitions as $p \xrightarrow{a} q$ for $(p, a, q) \in T$ and the extension of this to traces (written $p \xrightarrow{tr} q$) and the set of enabled actions of a process which is defined as: $next(p) = \{a \in Act \mid \exists\, q \bullet p \xrightarrow{a} q\}$.

To relate refinements in process algebras to those in a relational model, the methodology (as described in earlier papers [6,3,7,8]) is as illustrated in the following subsection. As an example in an untimed context we give the embedding of the trace semantics and trace preorder as follows.

3.1 Trace preorder

We first define the trace refinement relation.

Definition 9 *$\sigma \in Act^*$ is a trace of a process p if $\exists\, q \bullet p \xrightarrow{\sigma} q$. $\mathcal{T}(p)$ denotes the set of traces of p. The trace preorder is defined by $p \sqsubseteq_{tr} q$ iff $\mathcal{T}(q) \subseteq \mathcal{T}(p)$.* □

We then define a relational embedding of the Z data type, that is, define a data type (specifically define the finalisation operation) so as to facilitate the proof that data refinement equals the event based semantics. The choice of finalisation is taken so that we observe the characteristics of interest. Thus in the context of trace refinement we are interested in observing traces, but in that of failures refinement we need to observe more. So here possible traces lead to the single global value; impossible traces have no relational image.

Definition 10 (Trace embedding)
A Z data type $(State, Init, \{Op_i\}_{i \in J})$ has the following trace embedding into the relational model.

$$\mathsf{G} == \{*\}$$
$$\mathsf{State} == State$$
$$\mathsf{Init} == \{Init \bullet * \mapsto \theta State'\}$$
$$\mathsf{Op} == \{Op \bullet \theta State \mapsto \theta State'\}$$
$$\mathsf{Fin} == \{State \bullet (\theta State, *)\}$$

To distinguish between the different embeddings we denote the trace embedding of a data type A as $\mathsf{A}\,\big|_{tr}$. We drop the $\big|_{tr}$ if the context is clear. □

We then describe how to calculate the relevant LTS aspect from the Z data type. For example, for trace refinement what denotes traces (as in Definition 9) in the Z data type.

Definition 11 *The* traces *of a Z data type $(State, Init, \{Op_i\}_{i \in J})$ are all sequences $\langle i_1, \ldots, i_n \rangle$ such that*

$$\exists\, State' \bullet Init \fatsemi Op_{i_1} \fatsemi \ldots \fatsemi Op_{i_n}$$

We denote the traces of an ADT A by $\mathcal{T}(A)$. □

We now prove that data refinement equals the relevant event based definition of refinement:

Theorem 1. *With the trace embedding, data refinement corresponds to trace preorder. That is, when Z data types A and C are embedded as A and C,*

$$\mathsf{A}\,\big|_{tr} \sqsubseteq_{data} \mathsf{C}\,\big|_{tr} \quad \text{iff } \mathcal{T}(C) \subseteq \mathcal{T}(A)$$

Finally, we extract a characterisation of refinement as simulation rules on the operations of the Z data type. These are of course the rules for standard Z refinement but omitting applicability of operations, as used also e.g., in Event-B.

$$\mathsf{CInit} \subseteq \mathsf{AInit} \fatsemi \mathsf{R}$$
$$\mathsf{R} \fatsemi \mathsf{CFin} \subseteq \mathsf{AFin}$$
$$\forall\, i : I \bullet \mathsf{R} \fatsemi \mathsf{COp}_i \subseteq \mathsf{AOp}_i \fatsemi \mathsf{R}$$

We thus have the following conditions for the trace embedding.

Definition 12 (Trace simulations in Z)
A relation R on $AState \land CState$ is a trace downward simulation *from A to C if*

$$\forall\, CState' \bullet CInit \Rightarrow \exists\, AState' \bullet AInit \land R'$$
$$\forall\, i \in J \bullet \forall\, AState;\ CState;\ CState' \bullet R \land COp_i \Rightarrow \exists\, AState' \bullet R' \land AOp_i$$

The relation T on $AState \land CState$ is a trace upward simulation *from A to C if it is total on $CState$ and*

$$\forall\, AState';\ CState' \bullet CInit \land T' \Rightarrow AInit$$
$$\forall\, i : J \bullet \forall\, AState';\ CState;\ CState' \bullet (COp_i \land T') \Rightarrow (\exists\, AState \bullet T \land AOp_i)$$ □

4 Timed models

We now add in the consideration of time into our framework. We adapt our definitions of data type, refinement and simulations given above for the untimed case, and again consider different refinement relations based upon differing semantics.

4.1 A timed relational model

We do not consider internal or silent actions, as their consideration complicates some of the timing based issues and these are discussed below. The model we define is essentially that of timed automaton [13], and we make the following assumptions:

- We use a continuous time domain $\mathbb{R}^+$, although we could parameterize the theory by a time domain $\mathcal{T}$.
- We have a single global clock.
- We impose no constraints about Zeno behaviour or its absence.
- We have no internal events.
- We have no timelocks, time can always progress.
- *Time additivity:* If time can advance by a particular amount d in two steps, then it can also advance by d in a single step. This corresponds to the axiom **S1** in [13].
- *Time interpolation:* which is the converse of time additivity, that is, for any time progression there is an intermediate state at any instant during that progression. This corresponds to the axiom **S2** in [13].

Additional constraints can be placed as is done in many models. For example, *constancy of offers* [23] states that the progression of time does not introduce any new events as possibilities, nor allows the withdrawal of any offers. Another axiom which is commonly included is that of *time determinism*, that is, that if time evolves but no internal or other visible action takes place then the new state is uniquely determined.

Our relational model will then be constructed with the following in mind:

- We assume the specifications contain a reserved variable t (of type $\mathbb{R}^+$) representing the current time which is part of the local state *State*, which also serves as the relational local state State.
- Atomic operations can refer to t but not change it, that is, they are instantaneous.
- The time passing operation Op_τ advances t by τ as well as possibly having other effects.
- A variable t representing time is also added to the global state G. Since time can always progress, these time passing operations are total.

– The finalisation Fin maps t in State to t in G, that is, it makes time visible at the end of a computation.
– Programs are then, as before, finite sequences of time passing or atomic operations.

With this construction (which is just an embedding of time into our relational model) Definitions 1 and 2 define notions of data type and program, and Definitions 3 and 4 give us definitions of refinement and simulations for use on the timed relational model. Furthermore, the assumption of a global clock holds for both abstract and concrete systems, and simulation conditions on the given finalisation imply that the retrieve relation has to be the identity as far as time t is concerned.

4.2 A timed behavioural model

To relate the simulations with differing refinement preorders we need a timed semantics such as provided by timed automaton, e.g., [13], or the semantics for a timed process algebra, e.g., [20,23]. To do so we will augment our LTSs with visible time passing actions, and will not make a fundamental distinction between a transition due to time and one due to another atomic action as is done in some process algebraic semantics (i.e., we do not have two types of transitions). Similarly we do not, at this stage, make a distinction between visible time actions and other external actions as is done in, say, [23] (i.e., we do not have two types of actions or events), c.f. pp3 of [13].

Example 1. The automaton A in Figure 1 represents a system that places no timing constraints on the sequential ordering of the events a then b then *stop*. In this, and subsequent figures, we denote arbitrary time passing by the transition $\xrightarrow{d}$.

In Figure 1 B adapts the above to introduce a specific time delay of at least 1 time units (assumed here to be secs) between a and b. This system corresponds to the timed CSP behaviour of $a \xrightarrow{1} b \rightarrow stop$. A timeout can be modelled by allowing a state-change

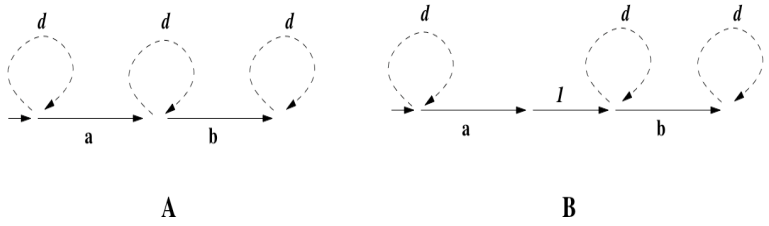

Fig. 1. Simple timed specifications

to occur at a certain point in time. Figure 2 corresponds to the timed CSP expression $(a \rightarrow stop) \overset{10}{\triangleright} (c \rightarrow stop)$. Note that in the presence of the requirement of constancy of offers it would be necessary to use an internal event to model the timeout at 10 secs; without such a requirement we can use a transition of 0 secs which represents a state change at that moment in time. □

Timed traces, which consist of a trace of action-time pairs (the time component recording the time of occurrence of the action), are often used as the representation of visible executions in a timed model (e.g., as in [13,20,23]). However, in order that we can reuse

results from Section 3 we take timed traces to be simply traces over the visible actions, which here includes time and other atomic actions. Thus a timed trace here will be of the form $\langle t_1, a_1, t_2, a_2, \ldots \rangle$, where the strict alternation can be assumed without loss of generality. In[13] a mapping is defined between the set of traces given as action-time pairs and those consisting of traces over time and other atomic actions, thus there is no loss of generality in our assumption of the form of timed traces.

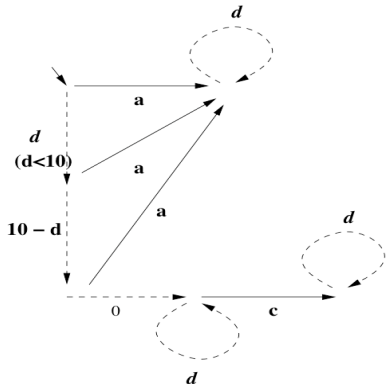

Fig. 2. A timeout

Example 2. The timed traces of A in Figure 1 include $\langle 4, a, 6.2, b \rangle$ and $\langle 0, 1, a, b, 4 \rangle$, those of B include $\langle 4, a, 1, b \rangle$, $\langle 3, a, 1, b, 7 \rangle$ etc. The relational model of B can be written as, where t_a is the time of occurrence of the action a:

$$
\begin{array}{l}
G == \mathbb{R}^+ \\
State == [t : \mathbb{R};\ x : \{0, 1, 2\};\ t_a : \mathbb{R}] \\
Init == \{t_0 : \mathbb{R}^+ \bullet t_0 \mapsto \langle\!\langle t = t_0, x = 0, t_a = 0 \rangle\!\rangle\} \\
Op_\tau == [\Delta State \mid x' = x \wedge t'_a = t_a \wedge t' = t + \tau] \\
a == [\Delta State \mid x = 0 \wedge x' = 1 \wedge t'_a = t \wedge t' = t] \\
b == [\Delta State \mid x = 1 \wedge x' = 2 \wedge t \geq t_a + 1 \wedge t' = t] \\
Fin == \lambda\, s : State \bullet s.t
\end{array}
$$

Timed trace preorder We now consider the first preorder: the timed trace preorder.

Definition 13 *Let Act be the atomic (non-time passing) actions, and* $t\text{–}Act = Act \cup \{Op_\tau \mid \tau \in \mathbb{R}^+\}$. *Let us denote the set of timed traces of* p *by* $\mathcal{T}_t(p) \subseteq (t\text{–}Act)^*$. *The timed trace preorder,* $\sqsubseteq_{t-tr}$, *is defined by* $p \sqsubseteq_{t-tr} q$ *iff* $\mathcal{T}_t(q) \subseteq \mathcal{T}_t(p)$. □

Example 3. Figure 1 defines A and B with $A \sqsubseteq_{t-tr} B$ but $B \not\sqsubseteq_{t-tr} A$. However, A and B have the same underlying *untimed* behaviour even though $A \neq_{t-tr} B$. □

Because of the construction of our timed relational model, and in particular, the consideration of timed actions as visible actions, the embeddings and correspondence given in Definitions 10 and 11 and Theorem 1 carry over directly to the timed case. Thus all that remains to be done is to derive the consequences for the simulation rules.

In fact, the derivations given in Definition 12 still hold providedt the quantification $\forall\, i : I$ includes quantification over the time passing events. All that remains is thus to articulate the precise consequences of this with regards to time. We consider these in turn.

Initialisation: the consequence here is, since the retrieve relation is the identity on time, the times at initialisation must be the same.

Non-time passing actions: the correctness condition falls apart into two parts: the usual correctness condition as in Definition 12 for the untimed aspects of the behaviour, with the additional consequence that

$$R \wedge \mathrm{pre}\, COp_i \Rightarrow \mathrm{pre}\, AOp_i$$

where the preconditions include reference to t – this in particular implies that in linked states, at the same time t, COp_i can only be enabled if AOp_i is.

Time passing actions: Since by assumption the time passing actions are total, the correctness condition in the blocking model thus becomes (for a downward simulation):

$$\forall\, \tau \in \mathbb{R}^+ \bullet \forall\, AState;\ CState;\ CState' \bullet R \wedge COp_\tau \Rightarrow \exists\, AState' \bullet R' \wedge AOp_\tau$$

if the time passing actions do not change the rest of the state space (i.e., we have time determinism), then this is vacuously satisfied.

Example 4. Figure 3 augments Figure 1 by the representation of the retrieve relation between the states of the two systems. With this retrieve relation it is easy to see that B is a timed-trace downward simulation of A. However, the reverse implication would not hold since

$$\mathrm{pre}\, AOp_i \Rightarrow \mathrm{pre}\, BOp_i$$

fails to hold at the state right after a in B. In a similar fashion the timed automaton

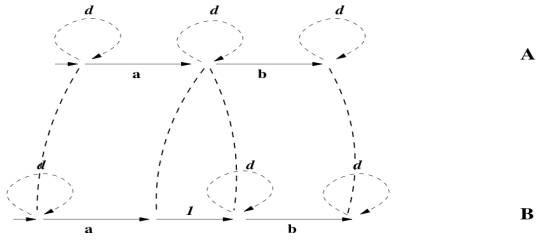

Fig. 3. A retrieve relation between timed specifications

corresponding to $P = (a \rightarrow stop)$ and $Q = (WAIT\ 2;\ a \rightarrow stop)$ have the same untimed behaviour and $P \sqsubseteq_{t-tr} Q$ but $Q \not\sqsubseteq_{t-tr} P$ since $\langle 1, a\rangle$ is a timed trace of P but not of Q. □

The above simulations agrees with those presented in [13], where Lynch and Vaandrager derive simulations for timed automata in a fashion similar to those presented in [12] for untimed automata (see also results in [8]). Specifically, they present a notion of *timed refinement* and the corresponding *timed forward simulation* and *timed backward*

simulation. As in the untimed case, their notion of timed refinement corresponds to timed trace preorder, and their timed simulations to those given by the simulations defined here.

Many of the results in [13] are derived via their untimed counterpart using the idea of a *closure automaton* which embeds a timed automaton into one with explicit time passing steps. Definitions and results on the closure automata can be translated directly to definitions and results on the underlying timed automata. In terms of our relational construction, such a closure corresponds to our basic definition since we consider time passing to be visible (in order that our results concerning simulations carry over to the timed case).

Our motivation for considering time passing to be a visible action here, though is twofold. Not only does it allow one to use the results of the preceding sections concerning simulations, but moreover, the view that time passing is indeed visible. Since we are less concerned with the parallel composition of two components in our relational framework, we do not need to stress the component-environment boundary and interface in a way that comes to the fore in, say, a process algebra. Thus we can either consider time to be visible, yet not under the control of the environment or even to be visible and under the control of an environment that will always offer time passing as one of its behaviours.

Completed timed trace preorder In an untimed setting, $\sigma \in Act^*$ is a completed trace of a process p if $\exists\, q \bullet p \xrightarrow{\sigma} q$ and $next(q) = \varnothing$. $\mathcal{CT}(p)$ denotes the set of completed traces of p. The completed trace preorder, $\sqsubseteq_{ctr}$, is defined by $p \sqsubseteq_{ctr} q$ iff $\mathcal{T}(q) \subseteq \mathcal{T}(p)$ and $\mathcal{CT}(q) \subseteq \mathcal{CT}(p)$.

To adapt the notion of the completed trace preorder we need to take completed traces as those that can do no more non-time passing actions (since time can always pass, we'd otherwise have no completed traces). Hence we define:

Definition 14 *σ is a completed timed trace of a process p if $\exists\, q \bullet p \xrightarrow{\sigma} q$ and $q \xrightarrow{tr}$ implies $tr \in \{Op_\tau\}^*$. $\mathcal{CT}_t(p)$ denotes the set of completed timed traces of p. The completed timed trace preorder, $\sqsubseteq_{t-ctr}$, is defined by $p \sqsubseteq_{t-ctr} q$ iff $\mathcal{T}_t(q) \subseteq \mathcal{T}_t(p)$ and $\mathcal{CT}_t(q) \subseteq \mathcal{CT}_t(p)$.* □

The basic relational embedding without time uses a global state that has been augmented with an additional element $\surd$, which denotes that the given trace is complete (i.e., no operation is applicable). Thus it uses the finalisation:

$$\mathsf{Fin} == \{State \bullet \theta State \mapsto *\} \cup \{State \mid (\forall\, i : J \bullet \neg \operatorname{pre} Op_i) \bullet \theta State \mapsto \surd\}$$

We have to adapt in a similar fashion in the presence of time, in particular, amend the finalisation so that we correctly record our completed timed traces. We thus change the global state to $\mathbb{R}^+ \times \{*, \surd\}$ and set

$$\begin{aligned}\mathsf{Fin} == &\{s : State \bullet s \mapsto (s.t, *)\} \\ &\cup \{State \mid (\forall\, i : I;\ \tau : \mathbb{R}^+ \bullet \neg \operatorname{pre}(Op_\tau \fatsemi Op_i)) \bullet \theta State \mapsto (\theta State.t, \surd)\}\end{aligned}$$

The effect on the downward simulation finalisation condition is that it becomes:

$$\begin{aligned}\forall\, AState;\ CState \bullet R \wedge \forall\, i : I;\ l\tau : \mathbb{R}^+ \bullet \neg \operatorname{pre}(COp_\tau \fatsemi COp_i) \Rightarrow \\ \forall\, i : I;\ \tau : \mathbb{R}^+ \bullet \neg \operatorname{pre}(AOp_\tau \fatsemi AOp_i)\end{aligned}$$

For an upward simulation, $\mathsf{CFin} \subseteq \mathsf{T} \fatsemi \mathsf{AFin}$ becomes

$$\begin{array}{l} \forall\, CState \bullet (\forall\, i : I;\ \tau : \mathbb{R}^+ \bullet \neg\, \mathrm{pre}(COp_\tau \fatsemi COp_i)) \Rightarrow \\ \qquad \exists\, AState \bullet\ T \wedge \forall\, i : I; \\ \qquad tau : \mathbb{R}^+ \bullet \neg\, \mathrm{pre}(AOp_\tau \fatsemi AOp_i) \end{array}$$

Furthermore, if one makes the additional assumption of constancy of offers, then these reduce to the untimed conditions [8], namely:

Downward simulations: $\mathsf{R} \fatsemi \mathsf{CFin} \subseteq \mathsf{AFin}$ is equivalent to

$$\forall\, AState;\ CState \bullet R \wedge \forall\, i : J \bullet \neg\, \mathrm{pre}\, COp_i \Rightarrow \forall\, i : J \bullet \neg\, \mathrm{pre}\, AOp_i$$

Upward simulations: $\mathsf{CFin} \subseteq \mathsf{T} \fatsemi \mathsf{AFin}$ is equivalent to

$$\forall\, CState \bullet \forall\, i : J \bullet \neg\, \mathrm{pre}\, COp_i \Rightarrow \exists\, AState \bullet\ T \wedge \forall\, i : J \bullet \neg\, \mathrm{pre}\, AOp_i$$

Those conditions, together with the above for non-time passing actions:

$$R \wedge \mathrm{pre}\, COp_i \Rightarrow \mathrm{pre}\, AOp_i$$

then give the required timed simulations for completed timed trace preorder.

Timed failure preorder Timed traces, as defined above, consist of traces where elements are either atomic or time passing actions. Our timed failures will be timed trace-refusals pairs, where refusals will be sets of actions which can be refused at the end of a trace. We thus use the following definition.

Definition 15 *$(\sigma, X) \in (t\text{–}Act)^* \times \mathbb{P}(Act)$ is a timed failure of a process p if there is a process q such that $p \stackrel{\sigma}{\longrightarrow} q$, and $next(q) \cap X = \varnothing$. $\mathcal{F}_t(p)$ denotes the set of timed failures of p. The timed failures preorder, $\sqsubseteq_{t\text{–}f}$, is defined by $p \sqsubseteq_{t\text{–}f} q$ iff $\mathcal{F}_t(q) \subseteq \mathcal{F}_t(p)$.*

Example 5. The timed automata corresponding to $Q = (WAIT\ 2;\ a \rightarrow stop)$ is a timed trace refinement, but not a timed failure refinement, of that corresponding to $P = (a \rightarrow stop)$. That is, $P \sqsubseteq_{t\text{–}tr} Q$ but $P \not\sqsubseteq_{t\text{–}f} Q$.

The timed failures of P include $(\langle d \rangle, \varnothing)$ for any time d, $(\langle d_1, a, d_2 \rangle, \{a\})$ for any times d_1, d_2. Those of Q include $(\langle 1 \rangle, \{a\})$ and $(\langle d_1, a, d_2 \rangle, \{a\})$ for any d_2 and any $d_1 \geq 2$. Thus $(\langle 1 \rangle, \{a\}) \in \mathcal{F}_t(Q)$ but this is not a failure of P. For the converse since we do not have timed trace refinement we cannot have timed failure refinement, for example, $(\langle 0.5, a \rangle, \varnothing) \in \mathcal{F}_t(P)$ but this is not a failure of Q. □

Since we have no timelocks, we have defined the timed refusals to be a subset of $\mathbb{P}(Act)$. We can adapt the relational embedding in the obvious way, and extracting the simulations we get those of the timed trace preorder (see Section 4.2 above) plus those due to the finalisation conditions:

Downward simulations: $\mathsf{R} \fatsemi \mathsf{CFin} \subseteq \mathsf{AFin}$ is equivalent to

$$\forall\, i : I \bullet \forall\, AState;\ CState \bullet R \wedge \mathrm{pre}\, AOp_i \Rightarrow \mathrm{pre}\, COp_i$$

Upward simulations: $\mathsf{CFin} \subseteq \mathsf{T} \fatsemi \mathsf{AFin}$ is equivalent to

$$\forall\, CState \bullet \exists\, AState \bullet \forall\, i : I \bullet\ T \wedge (\mathrm{pre}\, AOp_i \Rightarrow \mathrm{pre}\, COp_i)$$

However, note that the quantification over the state includes quantification over t, the variable representing time.

Example 6. Figure 4 represents (the principle parts of) a retrieve relation between the states of two systems. With this retrieve relation it is easy to see that B is a timed-failure downward simulation of A. □

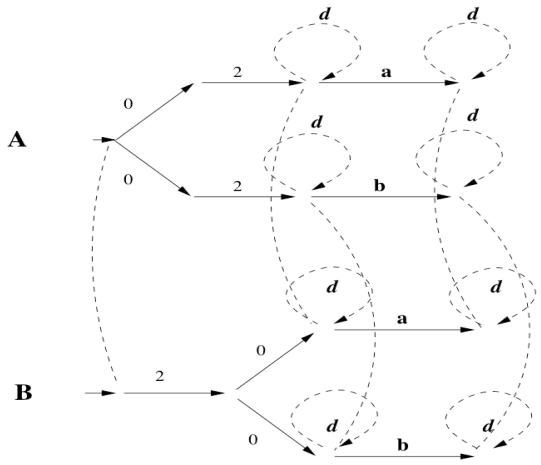

Fig. 4. A retrieve relation verifying a timed failure refinement

5 Discussion

In this paper we have discussed how time might be modelled relationally, and thus derived simulations for relational embeddings of a number of refinement preorders found in timed process algebras. Following the basic methodology defined above one can define further embeddings and preorders. For example, we can define a timed failure trace preorder which considers refusal sets not only at the end of a timed trace, but also between each action in a timed trace, and adapting the appropriate definitions is straightforward.

The timed failure preorder defined above determines the refusals at the *end* of a trace, in a fashion similar to the definition of refusal sets for an untimed automata or the failures preorder in CSP. This is in contrast to a number of failures models given for timed CSP, where refusals are determined throughout the trace rather than simply at the end. Thus these models are closer to a timed failure trace semantics as opposed to a timed failure semantics. The need to do this arises largely due to the treatment of internal events and, specifically, their urgency due to maximal progress under hiding.

There are a number of variants of these models, perhaps reflecting the fact that the presence of time has some subtle interactions with the underlying process algebra. They include the infinite timed failures model discussed in [23,15], which as the name suggests includes infinite traces in its semantics, as well as the timed failures-stability model of Reed and Roscoe [20]. A number of different models are developed in [18,19,20], and a hierarchy described in [17].

The common aspect of these models is that refusals are recorded *throughout* the trace rather than just at the end. Thus, for example, considering P, Q and R defined by:

$$P = a \rightarrow stop \quad Q = b \rightarrow stop \quad R = c \rightarrow stop$$

In untimed CSP we have

$$P \sqcap (Q \Box R) = (P \sqcap Q) \Box (P \sqcap R)$$

Furthermore, in a timed setting using the timed failure preorder defined in Section 4.2 this equivalence still holds. For example, the timed failures of both sides of this equivalence include $(\langle 1, b\rangle, \{a, b, c\})$, reflecting an execution where 1 time unit passes followed by the occurrence of b, and afterwards all events are refused. However, if refusals are recorded throughout the trace, then these for the RHS include $(\langle 1, b\rangle, [0, 1) \times \{c\})$, whereas those of the process on the LHS do not - this refusal representing an execution where c was refused over the time interval $[0, 1)$ and b occurred at time 1.

Indeed, using the timed failure preorder defined in Section 4.2, the above law of untimed CSP still remains valid, but this does not hold in any of the timed failures-stability models nor the infinite timed failures model [23,15]. The infinite timed failures model of Schneider et al [23,15] differs in two respects from the model in Section 4.2, namely the inclusion of infinite traces and the refusal information throughout the trace. The inclusion of infinite traces means that a better treatment can be given to divergence. Specifically, in a timed analysis a more precise treatment of an infinite sequence of internal events can be given, since an such an infinite sequence of internal events can now be classified as either: a timed divergence if they all occur at one instant in time; a Zeno divergence if they approach some finite time; or as a well-timed sequence if they take for ever for the whole sequence to occur [23]. Further, in timed CSP the first two possibilities are excluded from their notion of a well-timed process, thus nothing in timed CSP requires a treatment of divergence in the way that is necessary in untimed CSP.

The understanding of a failure (tr, X) in the infinite timed failures model is that after the trace tr the execution will eventually reach a point after which all events can be refused for the remainder of the execution. Even without consideration of refusals throughout the trace, this differs slightly from the model in Section 4.2, because there our traces observed the passing of time, and we do not need to say that the execution will *eventually* reach a point ..., since the quantification of 'eventually' is part of our timed traces. This model thus embeds the notion of constancy of offers since it requires that it will be possible to associate a timed refusal set of the form $[t, \infty) \times X$ with the execution. This restriction is not present in the model of Section 4.2 allowing time to change the events on offer.

It is argued in [23] that the inclusion of infinite traces in a timed failures model provides a more uniform treatment of unstable processes in the context of relating untimed and timed models of CSP, and indeed it allows a refinement theory (called *timewise refinement* [21,22]) to formally link the two models.

The alternative approaches use stability values, which denote the time by which any internal activity (including time) following the end of a trace must have ceased. Thus, the timed failures-stability model of Reed and Roscoe consists of triples (tr, X, α), where α is the stability. The use of stability can be seen in the following three processes:

$$P = a \rightarrow \mathit{stop} \quad Q = b \xrightarrow{3} \mathit{stop} \quad R = c \rightarrow \mathit{loop} \quad \text{where } \mathit{loop} = \mathit{wait}1;\ \mathit{loop}$$

Here $\xrightarrow{3}$ denotes a delay of 3 time units before control passing to the subsequent process, and *wait* 1 delays 1 time unit before terminating successfully. Then (see [23]) these three processes have identical timed failures but can be distinguished by different stability values. For example, P has $(\langle(1, a)\rangle, [0, 3) \times \{b\}, 1)$ as a behaviour, whereas Q has $(\langle(1, a)\rangle, [0, 3) \times \{b\}, 4)$ and R $(\langle(1, a)\rangle, [0, 3) \times \{b\}, \infty)$. Further details of stability values and how they are calculated are given in [20].

The need for refusals to be recorded during a trace in the above models arises from the urgency of internal events upon hiding in timed CSP. That arises ultimately from the consideration of the system/environment interface and the subsequent notion of *maximal progress*: that events occur when all participants are willing to engage in it. In the case of internal events, since the environment does not participate in them, they must occur when they become available. Now when an internal event occurs due to hiding, for example, as in the CSP process $P = (a \rightarrow stop) \setminus \{a\}$, the maximal progress condition implies such an internal event is *urgent*, that is, it occurs at the instance it becomes enabled.

One consequence of this urgency is that it forces negative premises in the transition rules for the hiding operator in timed CSP, specifically, time can only pass in $P \setminus A$ if no event from A is enabled in P.

This urgency also means we need more refusal information in order to maintain a *compositional semantics*. Reed and Roscoe give a number of examples of why hiding, urgency and a compositional semantics needs refusal information throughout the trace. One such example is the process $((a \rightarrow stop) \sqcap (wait\ 1;\ b \rightarrow stop)) \setminus \{a\}$. The urgency of the internal event upon hiding a means the non-deterministic choice is resolved in favour of $a \rightarrow stop$ and thus b cannot occur in a trace of this process. However, $\langle (1, b) \rangle$ is a trace of the process before hiding, meaning that more information is needed to regain compositionality. Further examples and discussion are given in [20].

However, in the context in which we began this discussion, namely a timed automata model, these issues are less relevant. Specifically, although a timed automata model can have internal events, a hiding operator is not defined, and therefore internal events do not have to be urgent. Furthermore, not all process algebraic models include urgency or maximal progress (see discussion in [16]). Additionally, one could argue that the urgency of internal events upon hiding is not entirely consistent with the informal semantics of untimed CSP. In particular, in untimed CSP the emphasis is on hidden events eventually occurring instantly: "a process exercises complete control over its internal events. ... [they] should not be delayed indefinitely once they are enabled" [23].

References

1. J. A. Bergstra, A. Ponse, and Scott A. Smolka, editors. *Handbook of Process Algebra*. Elsevier Science Inc., New York, NY, USA, 2001.
2. E.A. Boiten and J. Derrick. Modelling divergence in relational concurrent refinement. In M. Leuschel and H. Wehrheim, editors, *IFM 2009: Integrated Formal Methods*, volume 5423 of *LNCS*, pages 183–199. Springer Verlag, February 2009.
3. E.A. Boiten, J. Derrick, and G. Schellhorn. Relational concurrent refinement II: Internal operations and outputs. *Formal Aspects of Computing*, 21(1-2):65–102, 2009.
4. W.-P. de Roever and K. Engelhardt. *Data Refinement: Model-Oriented Proof Methods and their Comparison*. CUP, 1998.
5. J. Derrick and E. A. Boiten. *Refinement in Z and Object-Z*. Springer-Verlag, 2001.
6. J. Derrick and E.A. Boiten. Relational concurrent refinement. *Formal Aspects of Computing*, 15(1):182–214, November 2003.
7. J. Derrick and E.A. Boiten. More relational refinement: traces and partial relations. *Electronic Notes in Theoretical Computer Science*, 214:255–276, May 2008. Proceedings of REFINE 2008 (Turku, May 2008).
8. J. Derrick and E.A. Boiten. Relational concurrent refinement: Automata. *Electronic Notes in Theoretical Computer Science*, 259:21–34, December 2009. Proceedings of the

14th BCS-FACS Refinement Workshop (REFINE 2009). Extended version "Relational Concurrent Refinement III" submitted to Formal Aspects of Computing.
9. He Jifeng and C. A. R. Hoare. Prespecification and data refinement. In *Data Refinement in a Categorical Setting*, Technical Monograph, number PRG-90. Oxford University Computing Laboratory, November 1990.
10. He Jifeng, C. A. R. Hoare, and J. W. Sanders. Data refinement refined. In B. Robinet and R. Wilhelm, editors, *Proc. ESOP 86*, volume 213 of *Lecture Notes in Computer Science*, pages 187–196. Springer-Verlag, 1986.
11. C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.
12. N. Lynch and F. Vaandrager. Forward and backward simulations I.: untimed systems. *Information and Computation*, 121(2):214–233, 1995.
13. N. Lynch and F. Vaandrager. Forward and backward simulations Part II: timing-based systems. *Information and Computation*, 128(1):1–25, 1996.
14. R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.
15. M. W. Mislove, A. W. Roscoe, and S. A. Schneider. Fixed points without completeness. *Theoretical Computer Science*, 138(2):273–314, 1995.
16. X. Nicollin and J. Sifakis. An overview and synthesis on timed process algebras. In *Proceedings of the Real-Time: Theory in Practice, REX Workshop*, pages 526–548, London, UK, 1992. Springer-Verlag.
17. G.M. Reed. *A uniform mathematical theory for real-time distributed computing*. PhD thesis, 1988.
18. G.M. Reed and A. W. Roscoe. A timed model for communicating sequential processes. In Laurent Kott, editor, *ICALP*, volume 226 of *Lecture Notes in Computer Science*, pages 314–323. Springer, 1986.
19. G.M. Reed and A. W. Roscoe. A timed model for communicating sequential processes. *Theor. Comput. Sci.*, 58(1-3):249–261, 1988.
20. G.M. Reed and A. W. Roscoe. The timed failures-stability model for CSP. *Theoretical Computer Science*, 211(1-2):85–127, 1999.
21. S. Schneider. Timewise refinement for communicating processes. In *Proceedings of the 9th International Conference on Mathematical Foundations of Programming Semantics*, pages 177–214, London, UK, 1994. Springer-Verlag.
22. S. Schneider. Timewise refinement for communicating processes. *Sci. Comput. Program.*, 28(1):43–90, 1997.
23. S. Schneider. *Concurrent and Real Time Systems: The CSP Approach*. John Wiley & Sons, Inc., New York, NY, USA, 1999.
24. R. J. van Glabbeek. The linear time - branching time spectrum I. The semantics of concrete sequential processes. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, pages 3–99. North-Holland, 2001.
25. R.J. van Glabbeek. The linear time – branching time spectrum II; the semantics of sequential systems with silent moves (extended abstract).
26. J. C. P. Woodcock and J. Davies. *Using Z: Specification, Refinement, and Proof*. Prentice Hall, 1996.
27. Xiaodong Yuan, Jiajun Chen, and Guoliang Zheng. Duration calculus in COOZ. *SIGSOFT Softw. Eng. Notes*, 23, May 1998.