Technical Report 2008-001
A Testing Framework for Real-Time Specifications*

Chun Dai and Stefan D. Bruda
Department of Computer Science, Bishop's University
Sherbrooke, Quebec J1M 1Z7, Canada
email: {cdai|bruda}@cs.ubishops.ca

28 January 2008, revised 8 Oct 2008

Abstract

We propose a new semantic model for reasoning about real-time system specifications. Based on a theory of timed omega-final regular states, we present a framework of timed testing and two refinement timed preorders similar to De Nicola and Hennessy's may and must testing. We also provide alternative characterizations for these relations to show that the new preorders are extensions of the traditional preorders and to lay the basis for a unified logical and algebraic approach to conformance testing of real-time systems.

Keywords: formal methods, real-time systems, verification and validation, real-time software testing

1 Introduction

The development of hardware and software is getting more and more complex. How to guarantee validity and reliability is one of the most pressing problems nowadays [4, 8]. Among many theoretical methods for this, *conformance testing* [11, 14] is the most notable one for its succinctness and high automatization. Its aim is to check whether an implementation conforms to a given specification.

Formal system specifications [12], together with the implementations, can be mainly classified into two kinds: algebraic and logic. The first favors refinement, when a single algebraic formalism is equipped with a refinement relation to represent a system's specification and implementation [6, 15]. An implementation is validated correct if it refines its specification. Since it often defines the system transitionally, process algebrae [10], labelled transition systems [6], and finite automata are commonly used in this classification, with traditional refinement relations being either behavioural equivalences or preorders [6, 9]. A typical example is model-based testing [6]. The second approach to conformance testing prefers assertive constructs; different formalisms are used to describe the properties of the system specifications and implementations [2, 6]. Specifications are usually defined in a logical language while implementations are given in an operational notation. The semantics of assertions is to determine whether an implementation satisfies its specification. A typical example is model checking [2].

*This research was supported by the Natural Sciences and Engineering Research Council of Canada. Part of this research was also supported by Bishop's University.

The domain of conformance testing consists in reactive systems, which interact with their environment (also regarded as a reactive system). Often such systems are required to be *real time*, meaning that in addition to the correct order of events, they must satisfy constraints on delays separating certain events. A system that does not respond within our lifetime is obviously not useful, but many times we require a more precise time-wise characterization. *Real-time specifications* [12] are then used as the basis of conformance testing for such systems.

The aim of this paper is to develop a semantic theory for real-time system specifications featuring real-time transition systems modeling the behaviour of timed processes. Using a new theory of timed ω-final states as well as a timed testing framework based on De Nicola and Hennessy's may- and must-testing [9], we develop our timed may and must preorders that relate timed processes on the basis of their responses to timed tests. We also provide concise alternative characterizations of these two preorders. Our framework is as close to the original framework of (untimed) testing as possible, and is also as general as possible. While studies of real-time testing exist, they have mostly restricted the real-time domain to make it tractable; by contrast, our theory is general. As a consequence, it is perhaps not applicable directly in practice; however, it considers the whole domain with all its particularities. We believe that starting from a general theory is more productive than starting directly from some practically feasible (and thus restricted) subset of the issue.

2 Preliminaries

Preorders are reflexive and transitive relations. They are widely used as implementation relations comparing specifications and implementations. Preorders are easier to construct and analyze compared to equivalence relations, and once a preorder is established, an associated equivalence relation is immediate.

Our theory is based on some *action alphabet* A representing a set of actions excluding the internal action τ, and on a *time alphabet* L, which represent time values (often the set of strictly positive real numbers) and is ranged over by the set V of *time variables*. The set of *time clocks* C is a set of clocks (i.e, variables in V) associated to states. $\Phi\mathsf{C}$ is the set of time constraints over a set C of clocks. A *clock interpretation* for a set C is a mapping $\mathsf{C} \to \mathsf{L}$. *Clock progress* denotes the effect of time sequences that increase clock values. If $t > 0$ and k is a clock interpretation over C, in the clock interpretation $k' = k + t$ we have $k'(x) = k(x) + t$ for all clocks $x \in \mathsf{C}$. Clocks can be *reset* to zero.

A *time constraint* is also called *clock constraint* here, since we constrain the clocks of the states to constrain time between states. If x is a clock and c is a real number, then $x \sim c$ is a clock constraint, where $\sim \in \{\leq, <, =, \neq, >, \geq\}$. Clock constraints can be joined together either in conjunctions ($\wedge$) or disjunctions ($\vee$). That is, if x is a clock, the following is an admissible clock constraint: $x < 3 \vee x > 5$ ($\wedge$ is for multiple clocks).

A *time-event sequence* is a potentially infinite sequence of pairs of actions and time values, i.e., a member of $(\mathsf{A} \times \mathsf{L})^* \cup (\mathsf{A} \times \mathsf{L})^\omega$.

Timed automata [3] are based on the automata theory and introduce the notion of time constraints over their transitions. We define timed automata in terms of timed transition tables [3]: A timed automaton is a tuple (Σ, S, S_0, C, E), where Σ is a finite alphabet, S is a finite set of states, $S_0 \subseteq S$ is a set of start states, C is a finite set of clocks, and $E \subseteq S \times S \times \Sigma \times 2^C \times \Phi\mathsf{C}$ is the transition relation. A member $(s, s', a, \lambda, \phi)$ of the transition relation represents a transition form state s to state s' on input symbol a. The set λ gives the clocks to be reset with the transition, and ϕ is a clock constraint over C.

Using the theory of timed automata, we can transform a potentially infinite labelled transition system into a timed transition system. The difference between timed automata and timed transition systems is that the states, time intervals, and transitions in the later are not necessarily finite or even countable. A timed

transition system is essentially a labelled transition system extended with time values associated to actions. It is then a tuple $(S, (\mathsf{A} \times \mathsf{L}) \cup \{\tau\}, \rightarrow, s_0)$, where S is a countable, non-empty set of states; A is a set of actions with $\tau \notin \mathsf{A}$ representing a special, internal action; L is a set of time values (which we informally call time actions); $\rightarrow \subseteq S \times (\mathsf{A} \times \mathsf{L}) \cup \{\tau\} \times S$ is the timed transition relation; and $s_0 \in S$ is the initial state. Note that time can only be associated to visible actions.

3 Timed Processes, Timed Traces and Timed Languages

Labelled transition systems are used to model the behaviour of various processes. They serve as a semantic model for formal specification languages. Here, we define our notion of timed process based on timed transition systems.

Definition 3.1 For a set A of observable actions ($\tau \notin \mathsf{A}$), a set L of times values, and a set C of clocks with an associated set $\Phi\mathsf{C}$ of time constraints, a timed process is a tuple $((\mathsf{A} \times \mathsf{L}) \cup \{\tau\}, \mathsf{C}, S, \rightarrow, (s_0, c_0))$, where: S is a countable set of states; $p = (s, c) \in S$, where s is the location of the state and c is a clock interpretation over C[1]; $\rightarrow \subseteq S \times ((\mathsf{A} \times \mathsf{L}) \cup \{\tau\}) \times S \times \Phi\mathsf{C}$ is the transition relation (commonly, we use $p \xrightarrow[\Phi c]{(a,\delta)} p'$ instead of $(p, (a, \delta), p', \Phi c) \in \rightarrow$); (s_0, c_0) is the initial state[2].

The process picks its way from one state to the next state according to the transition relation. Whenever $(s, c) \xrightarrow[\Phi c]{(a,\delta)} (s', c')$, then the process performs a with delay δ; the delay causes the clocks to progress so that $c' = c + \delta$; the transition is enabled only if Φc holds in state (s, c).

Timed processes are distinguished from labelled transition systems in their treatment of traces (by associating time information with them). Normally a trace is described as a sequencing of the events or states, but not the delays between them. To add time to a trace, we add time information to the usual notion of trace (that contains actions only).

Definition 3.2 A *timed trace* over A, L, and $\Phi\mathsf{C}$ is a member of $(\mathsf{A} \times \mathsf{L} \times \Phi\mathsf{C})^* \cup (\mathsf{A} \times \mathsf{L} \times \Phi\mathsf{C})^\omega$, where A is a finite set of events, L is a set of time actions and $\Phi\mathsf{C}$ is a set of time constraints.

If both L and $\Phi\mathsf{C}$ are empty, the timed process is the same as a labelled transition system, and the timed trace is a normal trace. However, one of L or $\Phi\mathsf{C}$ could be empty and we still obtain a timed trace; this will be used later.

We will use the following relation: $p \xRightarrow[\Phi c]{(a,\delta)} p'$ iff $p = p_0 \xrightarrow{\tau} p_1 \xrightarrow{\tau} p_2 \xrightarrow{\tau} \cdots \xrightarrow{\tau} p_n \xrightarrow[\Phi c]{(a,\delta)} p'$ for some $n \geq 0$, and $p \xRightarrow{\varepsilon} p'$ iff $p = p_0 \xrightarrow{\tau} p_1 \xrightarrow{\tau} p_2 \xrightarrow{\tau} \cdots \xrightarrow{\tau} p_n = p'$ for some $n \geq 0$. By abuse of notation, we also write $p \xRightarrow{w} q$ whenever $w = (a_i, \delta_i, \Phi c_i)_{0<i\leq k}$ and $p \xRightarrow[\Phi c_1]{(a_1,\delta_1)} p_1 \xRightarrow[\Phi c_2]{(a_2,\delta_2)} p_2 \cdots \xRightarrow[\Phi c_k]{(a_k,\delta_k)} p_k = p'$.

Definition 3.3 Let $M = ((\mathsf{A} \times \mathsf{L}) \cup \{\tau\}, \mathsf{C}, S, \rightarrow, (s_0, c_0))$ be a timed process. A *timed path* $\pi(M)$ is a potentially infinite sequence $((s_{i-1}, c_{i-1}), (a_i, \delta_i, \Phi c_i), (s_i, c_i))_{0<i<k}$, where $(s_{i-1}, c_{i-1}) \xRightarrow[\Phi c_i]{(a_i,\delta_i)} (s_i, c_i)$, for all $0 < i \leq k$ with $a_i \in \mathsf{A}$, $\delta_i \in \mathsf{L}$, $\Phi c_i \in \Phi\mathsf{C}$.

[1] For simplicity we only consider one clock, as we only need one clock to establish our results. Generalizing to multiple clocks is however immediate.

[2] Whenever the transition relation is global and understood we can regard a state as the process whose initial state is the given state.

We use $|\pi|$ to refer to k, the length of π. If $|\pi| = \omega$, we say that π is *infinite*; otherwise, π is *finite*. A *deadlock* occurs when the process cannot move to another state. If $|\pi| \in \mathbb{N}$ and $(s_{|\pi|}, c_{|\pi|}) \not\rightarrow$ (i.e., $(s_{|\pi|}, c_{|\pi|})$ is a deadlock state), then the timed path π is called maximal. trace(π), the (timed) trace of π is defined as the sequence $(a_i, \delta_i, \Phi c_i)_{0 \leq i \leq |\pi|} \in (\mathsf{A} \times \mathsf{L} \times \Phi\mathsf{C})^*$.

We use $\Pi_f((s', c'))$, $\Pi_m((s', c'))$, $\Pi_I((s', c'))$ (or $\Pi_f(p')$, etc.) to denote the sets of all finite timed paths, all maximal timed paths, and all infinite timed paths starting from state $(s', c') \in S$ (or $p' \in S$), respectively. We also put $\Pi(p') = \Pi_f(p') \cup \Pi_m(p') \cup \Pi_I(p')$. The empty timed path π with $|\pi| = 0$ is symbolized by $()$ and its (always empty) trace by ε.

We can now introduce different languages for a timed process p.

Definition 3.4 The timed finite-trace language $L_f((s, c))$, timed maximal-trace (complete-trace) language $L_m((s, c))$, and timed infinite-trace language $L_I((s, c))$ of $p = (s, c)$ are

$$\begin{aligned} L_f((s,c)) &= \{\text{trace}(\pi) : \pi \in \Pi_f((s,c))\} \subseteq (\mathsf{A} \times \mathsf{L} \times \Phi\mathsf{C})^* \\ L_m((s,c)) &= \{\text{trace}(\pi) : \pi \in \Pi_m((s,c))\} \subseteq (\mathsf{A} \times \mathsf{L} \times \Phi\mathsf{C})^* \\ L_I((s,c)) &= \{\text{trace}(\pi) : \pi \in \Pi_I((s,c))\} \subseteq (\mathsf{A} \times \mathsf{L} \times \Phi\mathsf{C})^* \cup (\mathsf{A} \times \mathsf{L} \times \Phi\mathsf{C})^\omega. \end{aligned}$$

Then the set of initial actions of state $p' = (s', c')$ in process p if defined as follows: $I_p((s', c')) = \{(a, \delta, \Phi c) \in \mathsf{A} \times \mathsf{L} \times \Phi\mathsf{C} : \exists (s'', c'') : (s', c') \xrightarrow{\frac{(a,\delta)}{\Phi c}} (s'', c'')\}$.

3.1 Timed ω-languages and ω-final states

A timed ω-language is defined as a language that can be accepted by a timed ω-automaton [3]. A timed word over an alphabet Σ consists in a pair (σ, t), where $\sigma = \sigma_1\sigma_2\ldots$ is an infinite word over Σ and t is a time sequence (a sequence of time values). A timed language over Σ is then a set of timed words over Σ.

In our theory, timed ω-languages are defined slightly differently (notice however that there is a natural bijection between our definition and the original [3]), in order to reflect the use of such languages for system specification (where we consider that time passes only between actions) and also to simplify the presentation. A timed ω-language is then a set of time-event sequences (see Section 2) $\nu = \nu_1\nu_2\nu_3\cdots$. When the sequence is finite, the last element must be ω.

Timed ω-final states are the states which allow time-event sequences defined by a timed ω-regular language to be accepted by appearing infinitely often in the corresponding timed path. For untimed sequences and automata, the theory of ω-languages is not as simple as the theory of finite automata. The theory becomes even more complex when we move to timed traces where we have two notions of infinitude (event length and time length) which might not coincide. To alleviate this problem, we introduce ω-final states as a recurrence over the state sequence. The events between the states in a single recurrence are all the same, but the associated time intervals are not necessarily the same.

Consider for instance a very simple timed trace (over an empty set of time constraints) $(a, 1)(a, 1)(a, 1)\cdots$, whose infinite time-event sequence is $(a, 1)^\omega$. Another time trace might be $(a, 1)(a, 1/2)(a, 1/4)\cdots$, whose infinite time-event sequence is $(a, 1)(a, 1/2)(a, 1/4)\cdots$. If L has no positive lower-bound on the time length, then timed traces over L may exhibit Zeno behaviours, like our second example. Our design choice is to explicitly exclude such Zeno behaviours from the languages that we consider, that is, no sequence is allowed to show Zeno behaviour. The second trace in the example above is therefore not a valid trace (while the first one is). In other words, time progresses and must eventually grow past any constant value (this property is also called progress [3, 7]).

In all, we define the timed ω-regular-trace language as following:

Definition 3.5 The *timed ω-regular-trace language* of some process p is $L_\omega(p) = \{\text{trace}(\pi) : \pi \in \Pi_\omega(p)\} \subseteq (\mathsf{A} \times \mathsf{L} \times \Phi\mathsf{C})^* \cup (\mathsf{A} \times \mathsf{L} \times \Phi\mathsf{C})^\omega$, where $\Pi_\omega(p)$ contains exactly all the *ω-regular timed paths*. That is, ω-final states must occur infinitely often in any $\pi \in \Pi_\omega(p)$.

Divergence, the special notion of partially defined states (that may engage in an infinite internal computation), is important for the testing theory of reactive systems. State (s', c') of process p is *timed divergent*, denoted by $(s', c') \Uparrow_p$, if $\exists \pi \in \Pi_I((s', c')) : \text{trace}(\pi) = \varepsilon$. State (s', c') is called timed w-divergent (denoted by $(s', c') \Uparrow_p w$) for some $w = (a_i, \delta_i, \Phi c_i)_{0<i<k} \in (\mathsf{A} \times \mathsf{L} \times \Phi\mathsf{C})^* \cup (\mathsf{A} \times \mathsf{L} \times \Phi\mathsf{C})^\omega$ if one can reach a divergent state starting from (s', c') when executing a finite prefix of w, i.e., if $\exists l \in \mathbb{N}, (s'', c'') \in S : l \leq k, (s', c') \stackrel{w'}{\Longrightarrow} (s'', c''), s'' \Uparrow_p$, with $w' = (a_i, \delta_i, \Phi c_i)_{0<i<l}$. For convenience we write $L_D(p')$ for the divergence language of p', i.e., $L_D(p') = \{w \in (\mathsf{A} \times \mathsf{L} \times \Phi\mathsf{C})^* \cup (\mathsf{A} \times \mathsf{L} \times \Phi\mathsf{C})^\omega : p' \Uparrow_p w\}$.

Conversely, state (s', c') is timed convergent or timed w-convergent (denoted $(s', c') \Downarrow_p$ and $(s', c') \Downarrow_p w$, respectively) if it is not the case that $(s', c') \Uparrow_p$ and $(s', c') \Uparrow_p w$, respectively.

4 A Testing Theory

In this section we extend the testing theory of De Nicola and Hennessy [9] to timed testing. The traditional testing framework defines behavioural preorders that relate labelled transition systems according to their responses to tests [1]. Tests are used to verify the external interactions between a system and its environment. We use here timed processes as the basis for relating processes (and thus reasoning about timed specifications). Recall that our timed processes extend labelled transition systems not only with time information but also by their ability to consider infinite traces.

4.1 Timed tests and timed testing preorders

In our framework a test is a timed process where certain states are considered to be success states. In order to determine whether a system passes a test, we run the test in parallel with the system under test and examine the resulting finite or infinite computations until the test runs into a success state[3] (pass) or a deadlock state (fail). In addition, a set of ω-final states is used to compartmentalize the timed test into finite and infinite.

Definition 4.1 A *timed test* $((\mathsf{A} \times \mathsf{L}) \cup \{\tau\}, \mathsf{C}, T, \rightarrow_t, \Omega, (s_0^t, c_0^t), Suc)$ is a timed process $((\mathsf{A} \times \mathsf{L}) \cup \{\tau\}, \mathsf{C}, T, \rightarrow, (s_0, c_0))$ with the addition of a set $Suc \subseteq T$ of *success states* and a set $\Omega \subseteq T$ of ω-final states. Furthermore, $\mathsf{L} = \emptyset$ for tests and therefore $\rightarrow_t \subseteq S \times (\mathsf{A} \cup \{\tau\}) \times S \times \Phi\mathsf{C}$.

The transition relation differs from the original one because the test runs in parallel with the process under the test[4]. This latter process (called the implementation) features time sequences but no time constraints, while the test features only time constraints. It is meaningless to run the test by itself. If $\Phi\mathsf{C} = \emptyset$ which means there is no time constraint in the test, we call the test classical. The set of all timed tests is denoted by Γ.

Definition 4.2 A *partial computation* c with respect to a timed process p and a timed test t is a potentially infinite sequence $(\langle p_{i-1}, t_{i-1}\rangle \stackrel{(a_i, \delta_i)}{\underset{\Phi c_i}{\longmapsto}} R \langle p_i, t_i\rangle)_{0<i\leq k}$, where $k \in \mathbb{N} \cup \{\omega\}$, such that (1) $p_i \in P$ and $t_i \in T$ for

[3] Success states are deadlock states too, but we distinguish then as special deadlock states.

[4] Note however that the difference is syntactical only, as the transition relation for a timed process allows for an empty set L.

all $0 < i \leq k$, and (2) $(a_i, \delta_i) \in (\mathsf{A} \times \mathsf{L}) \cup \{\tau\}$ is taken from p, Φc_i is the time constraint (if any) taken from t and $R \in \{1, 2, 3\}$ for all $0 < i \leq k$.

The relation $\mapsto$ is defined by the following rules:

- $\langle p_{i-1}, t_{i-1}\rangle \xrightarrow[\Phi c_i]{(a_i,\delta_i)}{}_1 \langle p_i, t_i\rangle$ if $a_i = \tau$, $p_{i-1} \xrightarrow{\tau}_p p_i$, $t_{i-1} = t_i$, and $t_{i-1} \notin Suc$
- $\langle p_{i-1}, t_{i-1}\rangle \xrightarrow[\Phi c_i]{(a_i,\delta_i)}{}_2 \langle p_i, t_i\rangle$ if $a_i = \tau$, $p_{i-1} = p_i$, $t_{i-1} \xrightarrow{\tau}_t t_i$, and $t_{i-1} \notin Suc$
- $\langle p_{i-1}, t_{i-1}\rangle \xrightarrow[\Phi c_i]{(a_i,\delta_i)}{}_3 \langle p_i, t_i\rangle$ if $(a_i, \delta_i) \in \mathsf{A} \times \mathsf{L}$, $p_{i-1} \xrightarrow{(a_i,\delta_i)}_p p_i$, $t_{i-1} \xrightarrow[\Phi c_i]{(a_i,\delta_i)}_t t_i$, and $t_{i-1} \notin Suc$

The first expression in the definition of $\mapsto$ indicates that when the process under the test is executing an internal action from p_{i-1} to p_i, the test keeps its state. The second expression indicates that when the test is executing an internal action from t_{i-1} to t_i, the process under test keeps its state. The third expression indicates that when the action is not internal, the test and the process under test execute their respective action in parallel, and spend the same time while doing so. Moreover, the test also needs to check the time constraint.

If $k \in \mathbb{N}$ then c is finite, denoted by $|c| < \omega$; otherwise, it is infinite, i.e., $|c| = \omega$. The projection $\text{proj}_p(c)$ of c on p is defined as $(\langle p_{i-1}, (a_i, \delta_i), p_i\rangle)_{i \in I_p^c} \in \Pi(p)$, where $I_p^c = \{0 < i \leq k : R_i \in \{1, 3\}\}$. Similarly, the projection $proj_t(c)$ of c on t if defined as $(\langle t_{i-1}, (a_i, \delta_i, \Phi c_i), t_i\rangle)_{i \in I_t^c} \in \Pi(t)$, where $I_t^c = \{0 < i \leq k : R_i \in \{2, 3\}\}$.

Definition 4.3 A partial computation c is called *computation*, if it satisfies the following properties: (1) c is maximal, i.e., $k \in \mathbb{N}$ implies $p_k \not\xrightarrow{\tau}_p$, $t_k \not\xrightarrow{\tau}_t$ and $I_p(p_k) \cap I_t(t_k) = \emptyset$ (p_k and t_k cannot execute the same action), or the time sequence of p_k does not satisfy the time constraint of t_k; and (2) $k = \omega$ implies $\text{proj}_p(c) \in \Pi_I(p)$.

The set of all computations of p and t is denoted by $C(p, t)$.

Definition 4.4 Computation c is *successful* if $t_{|c|} \in Suc$ whenever $|c| \in \mathbb{N}$, and $\text{proj}_t(c) \in \Pi_\omega(t)$ whenever $|c| = \omega$.

Definition 4.5 *p may pass t*, denoted by $p \,\text{may}_{\mathbb{T}}\, t$, if there exists at least one successful computation $c \in C(p, t)$. Analogously, *p must pass t*, denoted by $p \,\text{must}_{\mathbb{T}}\, t$ if every computation $c \in C(p, t)$ is successful.

Intuitively, an infinite computation of process p and test t is successful if the test passes through a set of ω-final states infinitely often. Hence—in contrast with the untimed theory [3]—some infinite computations can be successful in our setting. Since timed processes and timed tests potentially exhibit nondeterministic behaviour, one may distinguish between the possibility and inevitability of success. This is captured in the following definition of the timed may and must preorders.

Definition 4.6 Let p and q be timed processes. Then, $p \sqsubseteq_{\mathbb{T}}^{may} q$ iff $\forall t \in \Gamma : p \,\text{may}_{\mathbb{T}}\, t \implies q \,\text{may}_{\mathbb{T}}\, t$; and $p \sqsubseteq_{\mathbb{T}}^{must} q$ iff $\forall t \in \Gamma : p \,\text{must}_{\mathbb{T}}\, t \implies q \,\text{must}_{\mathbb{T}}\, t$.

It is immediate that the relations $\sqsubseteq_{\mathbb{T}}^{may}$ and $\sqsubseteq_{\mathbb{T}}^{must}$ are preorders. They are defined analogously to the classical may and must preorders (which are based on labelled transition systems and restrict Γ to classical tests).

4.2 Alternative characterizations

We now present alternative characterizations of the timed may and must preorders. The characterizations are similar in style to other characterizations and provide the basis for comparing the existing testing theory to our timed testing. The first characterization is similar to the characterization of other preorders [1] and relates timed testing directly with the behaviour of processes.

Theorem 4.1 *1.* $p \sqsubseteq_{\mathbb{T}}^{may} q$ *iff* $L_f(p) \subseteq L_f(q)$ *and* $L_\omega(p) \subseteq L_\omega(q)$.

2. $p \sqsubseteq_{\mathbb{T}}^{must} q$ *iff for all* $w \in (\mathsf{A} \times \mathsf{L})^* \cup (\mathsf{A} \times \mathsf{L})^\omega$ *such that* $p \Downarrow w$ *it holds that:* (a) $q \Downarrow w$, (b) *if* $|w| < \omega$ *then* $\forall q' : q \stackrel{w}{\Longrightarrow} q'$ *implies* $\exists p' : p \stackrel{w}{\Longrightarrow} p'$ *and* $I_p(p') \subseteq I_q(q')$, *and* (c) *if* $|w| = \omega$ *then* $w \in L_\omega(q)$ *implies* $w \in L_\omega(p)$.

The second characterization is given in terms of timed trace inclusions, once more similarly to the characterization of other preorders [15]. Note that we are now concerned with $\sqsubseteq_{\mathbb{T}}^{must}$ only, as the simplest $\sqsubseteq_{\mathbb{T}}^{may}$ is already characterized in terms of timed traces in Theorem 4.1.

To introduce this result we need to introduce the notion of *pure nondeterminism*. We call a timed process p purely nondeterministic, if for all states p' of p, (a) $p' \stackrel{\tau}{\longrightarrow}_p$ implies $p' \stackrel{a}{\not\longrightarrow}_p$ and $|\{((a,\delta),p'') : p' \stackrel{(a,\delta)}{\longrightarrow}_p p''\}| = 1$. Note that every timed process p can be transformed to a purely nondeterministic timed process p', such that $L_f(p) = L_f(p')$, $L_D(p) = L_D(p')$, $L_m(p) = L_m(p')$, and $L_\omega(p) = L_\omega(p')$ by splitting every transition $p' \stackrel{(a,\delta)}{\longrightarrow}_p p''$ into two transitions $p' \stackrel{\tau}{\longrightarrow}_p p_{\langle p',(a,\delta),p''\rangle}$ and $p_{\langle p',(a,\delta),p''\rangle} \stackrel{(a,\delta)}{\longrightarrow}_p p''$, where $p_{\langle p',(a,\delta),p''\rangle}$ is a new, distinct state.

Theorem 4.2 *Let* p *and* q *be timed processes such that* p *is purely nondeterministic. Then* $p \sqsubseteq_{\mathbb{T}}^{must} q$ *iff all of the following hold:* (a) $L_D(q) \subseteq L_D(p)$, (b) $L_f(q) \setminus L_D(q) \subseteq L_f(p)$, (c) $L_m(q) \setminus L_D(q) \subseteq L_m(p)$, *and* (d) $L_\omega(q) \setminus L_D(q) \subseteq L_\omega(p)$.

With respect to finite traces, the characterizations of timed tests differ from the ones of classical preorders by the addition of time variables. We also need to refine the classical characterizations so as to capture the behaviour of timed may- and must-testing with respect to infinite traces. The proofs of the two characterization theorems rely on the properties of the following specific timed tests.

- For $w = (a_i, \delta_i)_{0<i\leq k} \in (\mathsf{A} \times \mathsf{L})^*$, let $t_w^{May,*} = (\mathsf{A} \cup \{\tau\}, \mathsf{C}, T, \rightarrow, \emptyset, 0, k)$, where $T = \{0, 1, \ldots, k\}$ and $\rightarrow = \{(i-1, a_i, i, c_i = \sum_{j=0}^{i} \delta_i) : 0 < i \leq k\}$.
- For $w = (a_i, \delta_i)_{0<i\leq k} \in (\mathsf{A} \times \mathsf{L},)^\omega$, let $t_w^{May,\omega} = (\mathsf{A} \cup \{\tau\}, \mathsf{C}, T, \rightarrow, T, 0, \emptyset)$, where $T = \mathbb{N}$, $\rightarrow = \{(i-1, a_i, i, c_i = \sum_{j=0}^{i} \delta_i) : i > 0\}$.
- For $w = (a_i, \delta_i)_{0<i\leq k} \in (\mathsf{A} \times \mathsf{L})^*$, let $t_w^{May,div} = (\mathsf{A} \cup \{\tau\}, \mathsf{C}, T, \rightarrow, \{k\}, 0, \emptyset)$, where $T = \{0, 1, \ldots, k\}$, $\rightarrow = \{(i-1, a_i, i, c_i = \sum_{j=0}^{i} \delta_i) : 0 < i \leq k\} \cup \{(k, \tau, k, \text{True})\}$.
- For $w = (a_i, \delta_i)_{0<i\leq k} \in (\mathsf{A} \times \mathsf{L})^*$, let $t_w^{\Downarrow,*} = (\mathsf{A} \cup \{\tau\}, \mathsf{C}, T, \rightarrow, \emptyset, 0, \{s\})$, where $T = \{0, 1, \ldots, k\} \cup \{s\}$, $\rightarrow = \{(i-1, a_i, i, c_i = \sum_{j=0}^{i} \delta_i) : 0 < i \leq k\} \cup \{(i, \tau, s, \text{True}) : 0 < i \leq k\}$.
- For $w = (a_i, \delta_i)_{0<i\leq k} \in (\mathsf{A} \times \mathsf{L})^\omega$, let $t_w^{\Downarrow,\omega} = (\mathsf{A} \cup \{\tau\}, \mathsf{C}, T, \rightarrow, \{s\}, 0, \{s\})$, where $T = \mathbb{N} \cup \{s\}$, $\rightarrow = \{(i-1, a_i, i, c_i = \sum_{j=0}^{i} \delta_i) : i > 0\} \cup \{(i, \tau, s, \text{True}) : i > 0\}$.
- For $w = (a_i, \delta_i)_{0<i\leq k} \in (\mathsf{A} \times \mathsf{L})^*$, let $t_w^{Must,*} = (\mathsf{A} \cup \{\tau\}, \mathsf{C}, T, \rightarrow, \emptyset, 0, \{s\})$, where $T = \{0, 1, \ldots, k\} \cup \{s\}$, $\rightarrow = \{(i-1, a_i, i, c_i = \sum_{j=0}^{i} \delta_i) : 0 < i \leq k\} \cup \{(i, \tau, s, \text{True}) : 0 \leq i < k\}$

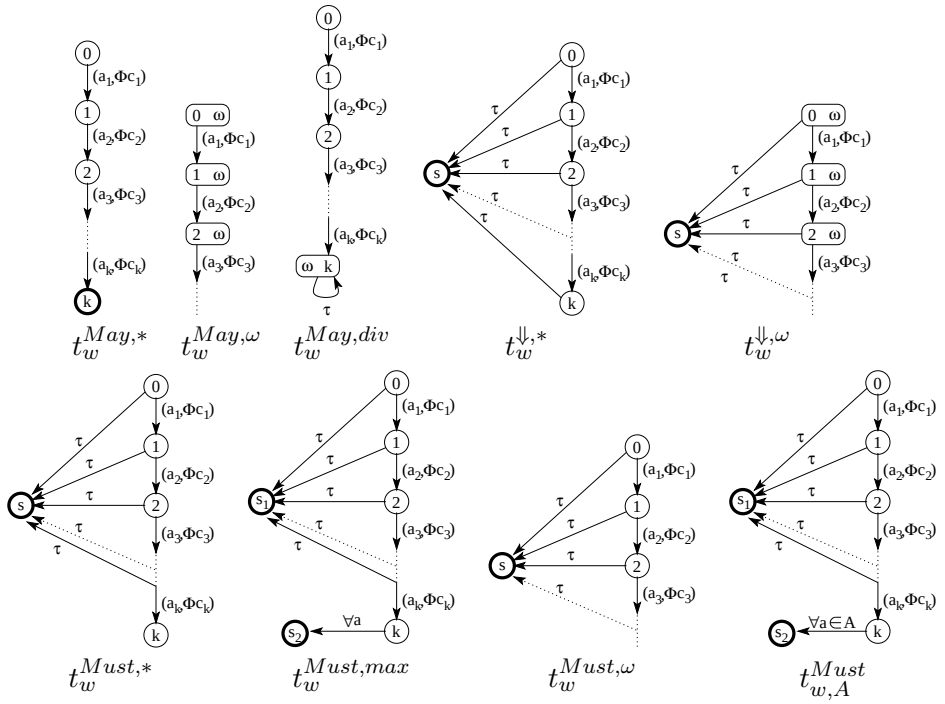

Figure 1: Timed tests used for the characterization of timed may and must preorders.

- For $w = (a_i, \delta_i)_{0<i\leq k} \in (\mathsf{A} \times \mathsf{L})^*$, let $t_w^{Must,max} = (\mathsf{A} \cup \{\tau\}, \mathsf{C}, T, \rightarrow, \emptyset, 0, \{s_1, s_2\})$, where $T = \{0, 1, \ldots, k\} \cup \{s_1, s_2\}$, $\rightarrow = \{(i-1, a_i, i, c_i = \sum_{j=0}^{i} \delta_i) : 0 < i \leq k\} \cup \{(i, \tau, s_1, \text{True}) : 0 \leq i < k\} \cup \{(k, a, s_2, \text{True}) : (a, \Phi c) \in \mathsf{A} \times \Phi\mathsf{C}$.
- For $w = (a_i, \delta_i)_{0<i\leq k} \in (\mathsf{A} \times \mathsf{L})^\omega$, let $t_w^{Must,\omega} = (\mathsf{A} \cup \{\tau\}, \mathsf{C}, T, \rightarrow, \emptyset, 0, \{s\})$, where $T = \mathbb{N} \cup \{s\}$, $\rightarrow = \{(i-1, a_i, i, c_i = \sum_{j=0}^{i} \delta_i) : i > 0\} \cup \{(i, \tau, s, \text{True}) : i \in \mathbb{N}\}$.
- For $w = (a_i, \delta_i)_{0<i\leq k} \in (\mathsf{A} \times \mathsf{L})^*$ and $A \subseteq \mathsf{A}$, let $t_{w,A}^{Must} = (\mathsf{A} \cup \{\tau\}, \mathsf{C}, T, \rightarrow, \emptyset, 0, \{s_1, s_2\})$, where $T = \{0, 1, \ldots, k\} \cup \{s_1, s_2\}$, $\rightarrow = \{(i-1, a_i, i, c_i = \sum_{j=0}^{i} \delta_i) : 0 < i \leq k\} \cup \{(i, \tau, s_1, \text{True}) : 0 \leq i < k\} \cup \{(k, a, s_2, \text{True}) : a \in A\}$.

These tests are depicted graphically in Figure 1. In the figure ω-final states are marked by the symbol ω and success states are distinguished from regular states by thick borders. Intuitively, while timed tests $t_w^{May,*}$ and $t_w^{May,\omega}$ test for the presence of a finite and infinite trace w, respectively, timed tests $t_w^{May,div}$, $t_w^{\Downarrow,*}$, and $t_w^{\Downarrow,\omega}$ are capable of detecting divergent behaviour when executing trace w. These are "presence" tests, that check whether a trace w (finite or infinite) exists in the implementation. Timed tests $t_w^{Must,*}$, $t_w^{Must,max}$, and $t_w^{Must,\omega}$ test for the absence of the finite trace, maximal trace, and ω-state trace (i.e., trace that goes through infinite occurrences of ω-final states) w, respectively. Timed must-testing is a little bit tricky, since we cannot feasibly check all the possible traces or computations exhaustively (as we need to do according to the definition of must testing). So we think the other way around: We assume one "failure trace," which does not satisfy the test and leads to failure. If there exists at least one such failure trace, then the test fails. On the other hand, if we cannot find the failure trace in the implementation, the test succeeds. We then test the absence of this trace in must-testing. Finally, timed test $t_{w,A}^{Must}$ is capable of comparing the initial action sets of states reached when executing trace w with respect to a subset $A \subseteq \mathsf{A}$. Note that we use the tightest time constraint possible in our test. We denote $c_i = \sum_{j=0}^{i} \delta_i$ by Φc_i in what follows (and also in Figure 1).

Our specific timed tests satisfy the following desired properties:

Lemma 4.3 *1. Let $w \in (\mathsf{A} \times \mathsf{L})^*$. Then, $w \in L_f(p)$ iff p may$_{\mathbb{T}}$ $t_w^{May,*}$.*

2. Let $w \in (\mathsf{A} \times \mathsf{L})^\omega$. Then $w \in L_\omega(p)$ iff p may$_{\mathbb{T}}$ $t_w^{May,\omega}$.

3. Let $w \in (\mathsf{A} \times \mathsf{L})^*$. Then, $w \in L_\omega(p)$ iff p may$_{\mathbb{T}}$ $t_w^{May,div}$.

4. Let $w \in (\mathsf{A} \times \mathsf{L})^*$. Then, $p \Downarrow w$ iff p must$_{\mathbb{T}}$ $t_w^{\Downarrow,*}$.

5. Let $w \in (\mathsf{A} \times \mathsf{L})^* \cup (\mathsf{A} \times \mathsf{L})^\omega$. Then, $p \Downarrow w$ iff p must$_{\mathbb{T}}$ $t_w^{\Downarrow,\omega}$.

6. Let $w \in (\mathsf{A} \times \mathsf{L})^*$ such that $p \Downarrow w$. Then, $w \notin L_f(p)$ iff p must$_{\mathbb{T}}$ $t_w^{Must,*}$.

7. Let $w \in (\mathsf{A} \times \mathsf{L})^*$ such that $p \Downarrow w$. Then, $w \notin L_m(p)$ iff p must$_{\mathbb{T}}$ $t_w^{Must,max}$.

8. Let $w \in (\mathsf{A} \times \mathsf{L})^\omega$ such that $p \Downarrow w$. Then, $w \notin L_\omega(p)$ iff p must$_{\mathbb{T}}$ $t_w^{Must,\omega}$.

Proof. The proofs are simple analyses of the potential computations arising when running the timed tests in lock-step (to a deadlock or successful state) with arbitrary timed processes. Let $w = (a_i, \delta_i)_{0<i\leq k}$ for some $k \in \mathbb{N} \cup \{\omega\}$.

- Item 1, $\Rightarrow$: $w \in L_f(p)$, and thus $p_0 \stackrel{(a_1,\delta_1)}{\Longrightarrow} p_1 \stackrel{(a_2,\delta_2)}{\Longrightarrow} \cdots \stackrel{(a_k,\delta_k)}{\Longrightarrow} p_k$ (Definition 3.4). On the other hand, $t_0^{May,*} \stackrel{(a_1,\delta_1)}{\underset{\Phi c_1}{\Longrightarrow}} t_1^{May,*} \stackrel{(a_2,\delta_2)}{\underset{\Phi c_2}{\Longrightarrow}} \cdots \stackrel{(a_k,\delta_k)}{\underset{\Phi c_k}{\Longrightarrow}} t_k^{May,*}$ (definition of $t_w^{May,*}$ including the form of Φc_i). Therefore, $(\langle p_{i-1}, t_{i-1}^{May,*}\rangle \stackrel{(a_i,\delta_i)}{\underset{\Phi c_i}{\longmapsto}} R \langle p_i, t_i^{May,*}\rangle)_{0<i\leq k}$, so w is the trace of a potential computation c for both p and $t_w^{May,*}$. In fact w is even the trace of a computation of p and $t_w^{May,*}$ (indeed, $t_k^{May,*} \stackrel{\tau}{\not\longrightarrow}$ and $I_p(p_k) \cap I_t(t_k^{May,*}) = \emptyset$), and is further the trace of a successful computation (since $t_k^{May,*} \in Suc$). It then follows that p may$_{\mathbb{T}}$ $t_w^{May,*}$.

  $\Leftarrow$: Given that p may$_{\mathbb{T}}$ $t_w^{May,*}$, we have a successful computation c of p and $t_w^{May,*}$. That is, $(\langle p_{i-1}, t_{i-1}^{May,*}\rangle \stackrel{(a_i,\delta_i)}{\underset{\Phi c_i}{\longmapsto}} R \langle p_i, t_i^{May,*}\rangle)_{0<i\leq k}$, $t_k^{May,*} \stackrel{\tau}{\not\longrightarrow}$, $I_p(p_k) \cap I_t(t_k^{May,*}) = \emptyset$, and $t_w^{May,*} = t_k^{May,*} \in Suc$. By a reverse argument we conclude then that $w \in L_f(p)$ ($t_0^{May,*} \stackrel{(a_1,\delta_1)}{\underset{\Phi c_1}{\Longrightarrow}} t_1^{May,*} \stackrel{(a_2,\delta_2)}{\underset{\Phi c_2}{\Longrightarrow}} \cdots \stackrel{(a_k,\delta_k)}{\underset{\Phi c_k}{\Longrightarrow}} t_k^{May,*}$, then $p_0 \stackrel{(a_1,\delta_1)}{\Longrightarrow} p_1 \stackrel{(a_2,\delta_2)}{\Longrightarrow} \cdots \stackrel{(a_k,\delta_k)}{\Longrightarrow} p_k$, and thus $w \in L_f(p)$). Items 2 and 3 are proven similarly.

- Item 4, $\Rightarrow$: Assume that p must$_{\mathbb{T}}$ $t_w^{\Downarrow,*}$ does not hold. However, the trace w passes $t_w^{\Downarrow,*}$ (by the definition of $t_w^{\Downarrow,*}$), only divergence can cause the test to fail. So for some $0 < l \leq k$ there exists one trace $p_0 \stackrel{(a_1,\delta_1)}{\Longrightarrow} p_1 \stackrel{(a_2,\delta_2)}{\Longrightarrow} \cdots \stackrel{(a_l,\delta_l)}{\Longrightarrow} p_l \stackrel{\tau}{\longrightarrow} p_l \cdots$ which means that $p \Uparrow w$, a contradiction. So it must be that p must$_{\mathbb{T}}$ $t_w^{\Downarrow,*}$.

  $\Leftarrow$: Assume that $p \Uparrow w$. Then for some $0 < l \leq k$ there exists one trace $p_0 \stackrel{(a_1,\delta_1)}{\Longrightarrow} p_1 \stackrel{(a_2,\delta_2)}{\Longrightarrow} \cdots \stackrel{(a_l,\delta_l)}{\Longrightarrow} p_l \stackrel{\tau}{\longrightarrow} p_l \cdots$ which fails the test $t_w^{\Downarrow,*}$. This contradicts the condition p must$_{\mathbb{T}}$ $t_w^{\Downarrow,*}$ and so it must be that $p \Downarrow w$. Item 5 is proven similarly.

- Item 6, $\Rightarrow$: Assume that p must$_{\mathbb{T}}$ $t_w^{Must,*}$ does not hold. According to the definition of $t_w^{Must,*}$, there are two ways for p to fail the test: Either $p_0 \stackrel{(a_1,\delta_1)}{\Longrightarrow} p_1 \stackrel{(a_2,\delta_2)}{\Longrightarrow} \cdots \stackrel{(a_i,\delta_i)}{\Longrightarrow} p_i \stackrel{\tau}{\longrightarrow} p_i \cdots$, or $p_0 \stackrel{(a_1,\delta_1)}{\Longrightarrow} p_1 \stackrel{(a_2,\delta_2)}{\Longrightarrow} \cdots \stackrel{(a_k,\delta_k)}{\Longrightarrow} p_k$. These contradict the conditions $p \Downarrow w$ or $w \notin L_f(p)$, respectively.

  $\Leftarrow$: Assume that $w \in L_f(p)$. By the definition of $t_w^{Must,*}$, w fails to pass this test. This contradicts the condition that p must$_{\mathbb{T}}$ $t_w^{Must,*}$. Items 7 and 8 are proven similarly. □

The proof of Theorem 4.1 relies extensively on these intuitive properties of timed tests. Notice that the usage of ω-state tests (that is, tests that accept based on an acceptance family, not only on Suc)—even when discussing finite-state timed processes—is justified by our view that timed tests represent the arbitrary, potentially irregular behaviour of the unknown real-time environment.

Proof of Theorem 4.1 Item 1 of the theorem is fairly immediate. For the $\Rightarrow$ direction we distinguish the following cases: $w \in L_f(p)$ implies that p may$_{\mathbb{T}}$ $t_w^{May,*}$. Since $p \sqsubseteq_{\mathbb{T}}^{may} q$ it follows that q may$_{\mathbb{T}}$ $t_w^{May,*}$ and thus $w \in L_f(q)$. $w \in L_\omega(p)$] has two sub-cases: (a) If $|w| = \omega$, then p may$_{\mathbb{T}}$ $t_w^{May,\omega}$. Since $p \sqsubseteq_{\mathbb{T}}^{may} q$ it follows that q may$_{\mathbb{T}}$ $t_w^{May,\omega}$ and thus $w \in L_\omega(q)$. (b) If $|w| < \omega$, then p may$_{\mathbb{T}}$ $t_w^{May,div}$. Since $p \sqsubseteq_{\mathbb{T}}^{may} q$ it follows that q may$_{\mathbb{T}}$ $t_w^{May,div}$ and thus $w \in L_\omega(q)$.

We go now to the $\Leftarrow$ direction for Item 1. Let t be any timed process such that p may$_{\mathbb{T}}$ t, i.e., there exists a successful computation $c \in C(p,t)$ with $w = \text{trace}(\text{proj}_p(c)) = \text{trace}(\text{proj}_t(c))$. If $|w| = \omega$, then $w \in L_\omega(p)$ and thus $w \in L_\omega(q)$ (since $L_\omega(p) \subseteq L_\omega(q)$). It follows that we can construct a successful computation $c' \in C(q,t)$ such that $w = \text{trace}(\text{proj}_q(c')) = \text{trace}(\text{proj}_t(c'))$ and $\text{proj}_t(c') = \text{proj}_t(c)$. It follows that q may$_{\mathbb{T}}$ t and therefore $p \sqsubseteq_{\mathbb{T}}^{may} q$. If $|w| < \omega$, we can split the proof into two cases: either $w \in L_f(p)$ or $w \in L_\omega(p)$. We can then establish that q may$_{\mathbb{T}}$ t as above.

On to Item 2 now. For the $\Rightarrow$ direction we have that $p \sqsubseteq_{\mathbb{T}}^{must} q$, $w \in (\mathsf{A} \times \mathsf{L})^* \cup (\mathsf{A} \times \mathsf{L})^\omega$ such that $p \Downarrow w$. Then p must$_{\mathbb{T}}$ $t_w^{\Downarrow,*}$ or p must$_{\mathbb{T}}$ $t_w^{\Downarrow,\omega}$ (Lemma 4.3), then q must$_{\mathbb{T}}$ $t_w^{\Downarrow,*}$ or q must$_{\mathbb{T}}$ $t_w^{\Downarrow,\omega}$ (Definition 4.6), thus $q \Downarrow w$ (Lemma 4.3). We further distinguish two cases, depending on whether $|w| = \omega$ or not:

If $|w| < \omega$, let $q \stackrel{w}{\Longrightarrow} q'$ for some q', i.e., $w \in L_f(q)$. Assume that there is no p' such that $p \stackrel{w}{\Longrightarrow} p'$ and $I_p(p') \subseteq I_q(q')$. Suppose that $p \stackrel{w}{\not\Longrightarrow}$ i.e., $w \notin L_f(p)$. Then p must$_{\mathbb{T}}$ $t_w^{Must,*}$ (Lemma 4.3), so q must$_{\mathbb{T}}$ $t_w^{Must,*}$ (Definition 4.6). However, q must$_{\mathbb{T}}$ $t_w^{Must,*}$ does not hold (contrapositive of Lemma 4.3), a contradiction. Suppose now that $p \stackrel{w}{\Longrightarrow}$. Let then $X = \{(a,\delta) \in I_p(p') : p \stackrel{w}{\Longrightarrow} p'\} \neq \emptyset$. Since $I_p(p') \not\subseteq I_q(q')$ (assumption), for every $A \in X$ there exists an $(a,\delta) \in A \setminus I_q(q')$. Let B be the set of all such actions a (ignoring the time actions). It is then immediate then that p must$_{\mathbb{T}}$ $t_{w,B}^{Must}$ (by the construction of $t_{w,B}^{Must}$); however, it is not the case that q must$_{\mathbb{T}}$ $t_{w,B}^{Must}$ (since $q' \stackrel{(a,\delta)}{\not\Longrightarrow}$ for any $(a,\delta) \in (B,\mathsf{L})$). This contradicts the assumption that $p \sqsubseteq_{\mathbb{T}}^{must} q$.

If on the other hand $|w| = \omega$, assume that $w \notin L_\omega(p)$. Then p must$_{\mathbb{T}}$ $t_w^{Must,\omega}$ (Definition 4.6) and thus $w \notin L_\omega(q)$ (Lemma 4.3). This contradicts with $w \in L_\omega(q)$ (given).

Finally, for the $\Leftarrow$ direction of Item 2, let t be any timed process such that q must$_{\mathbb{T}}$ t does not hold, i.e., there exists an unsuccessful computation $c = (\langle q_{i-1}, t_{i-1}\rangle, (a_i, \delta_i, \Phi c_i), \langle q_i, t_i\rangle)_{0<i\leq k} \in C(q,t)$ (Definition 4.5). Let $w = \text{trace}(\text{proj}_p(c)) = \text{trace}(\text{proj}_t(c))$.

Assume that $p \Uparrow w$. We can then construct an unsuccessful, infinite computation c' which resembles c until p can engage in its timed divergent computation and then we force t not to contribute anymore. Then $\text{proj}_p(c') \in \Pi_\omega(p)$ and $\text{proj}_t(c') \notin \Pi_\omega(t)$ (because $|proj_p(c')| < \omega$).This implies that p must$_{\mathbb{T}}$ t does not hold (Definition 4.5) and thus $p \sqsubseteq_{\mathbb{T}}^{must} q$ (since q must$_{\mathbb{T}}$ t does not hold, by the contrapositive of Definition 4.6).

Assume now that $p \Downarrow w$, i.e., $w \notin L_D(p)$. We have again two cases depending on whether $|c| < \omega$ or not.

Whenever $|c| < \omega$, (a) $w \in L_f(q)$, $q \stackrel{w}{\Longrightarrow} q'$ for some q' and $t_k \neq Suc$ by definition of $t_w^{Must,*}$, (b) $q_k \stackrel{\tau}{\not\longrightarrow}$, $t_k \stackrel{\tau}{\not\longrightarrow}$, $I_q^c(q_k) \cap I_t^c(t_k) = \emptyset$ by definition of $t_w^{Must,max}$; and (c) $\exists p' : p \stackrel{w}{\Longrightarrow} p', I_p^c(p') \subseteq I_q^c(q')$ by condition 2(b). By observations (a)–(c) we have a finite computation $c' = (\langle p_{i-1}, t'_{i-1}\rangle, (a_i, \delta_i, \Phi c_i), \langle p_i, t'_i\rangle)_{0<i\leq l} \in C(p,t)$ with $\text{proj}_t(c') = \text{proj}_t(c)$ and $\langle p_l, t_l\rangle = \langle p'', t_k\rangle$, where $p' \stackrel{\varepsilon}{\Longrightarrow} p''$ for some $p'' \stackrel{\tau}{\not\longrightarrow}_p$. Note that such a p'' must exist since $p \Downarrow w$. Then $I_p^c(p'') \subseteq I_p^c(p')$, definition of c' and p'', and observations (a) and (b) above imply that $I_p^c(p'') \cap I_t^c(t_k) \subseteq I_q^c(q') \cap I_t^c(t'_l) \subseteq I_q^c(q_k) \cap I_t^c(t_l)$; thus c' cannot be extended.

Since $t'_l = t_k \notin Suc$, c' is unsuccessful, so p must$_{\mathbb{T}}$ t does not hold.

Whenever $|c| = \omega$, q must$_{\mathbb{T}}$ $t_w^{Must,\omega}$ does not hold. It follows that $w \in L_\omega(q)$, and thus $w \in L_\omega(p)$ (given). So p must$_{\mathbb{T}}$ $t_w^{Must,\omega}$ does not hold either (contrapositive of Lemma 4.3). In all, $p \sqsubseteq_{\mathbb{T}}^{must} q$, as desired. □

The proof of Theorem 4.2 also relies on the properties of the timed tests introduced in Lemma 4.3.

Proof of Theorem 4.2 For the $\Rightarrow$ direction, assume that $p \sqsubseteq_{\mathbb{T}}^{must} q$ and let $w \in (\mathsf{A} \times \mathsf{L})^* \cup (\mathsf{A} \times \mathsf{L})^\omega$. Then,

(1) $w \in L_D(q)$ implies $q \Uparrow w$, so it is not the case that q must$_{\mathbb{T}}$ $t_w^{\Downarrow,\omega}$ (Lemma 4.3(5)). Therefore it is not the case that p must$_{\mathbb{T}}$ $t_w^{\Downarrow,\omega}$ (since $p \sqsubseteq_{\mathbb{T}}^{must} q$), so $p \Uparrow w$, or $w \in L_D(p)$, as desired.

(2) $w \in L_f(q) \setminus L_D(q)$ implies $q \Downarrow w$ and thus $p \Downarrow w$ (same as (1) but using Lemma 4.3(4)). In addition, it is not the case that q must$_{\mathbb{T}}$ $t_w^{Must,*}$ (Lemma 4.3(6)) and thus p must$_{\mathbb{T}}$ $t_w^{Must,*}$ does not hold (since $p \sqsubseteq_{\mathbb{T}}^{must} q$). Therefore, $w \in L_f(p)$, again as desired.

The proofs of (3) and (4) are the same as the proof of (2) using Lemma 4.3(7) and Lemma 4.3(8), respectively.

On to the $\Leftarrow$ direction now. We assume that (1), (2), (3), and (4) hold. We further assume that there exists a timed test t such that q must$_{\mathbb{T}}$ t does not hold (if such a test does not exist then $p \sqsubseteq_{\mathbb{T}}^{must} q$ for any process p). Thus there exists an unsuccessful computation $c = (\langle q_{i-1}, t_{i-1}\rangle(a_i, \delta_i)\langle q_i, t_i\rangle)_{0<i\leq k} \in C(q,t)$, with $w = \text{trace}(\text{proj}_q(c)) = trace(\text{proj}_t(c))$.

If $p \Uparrow w$ then construct an unsuccessful, infinite computation c' which resembles c until p can engage in its divergent computation, at which point t can be forced to stop contributing to c'. Thus $q \Uparrow w$ and it is not the case that p must$_{\mathbb{T}}$ t.

If $p \Downarrow w$, $|c| < \omega$, and $t_k \notin Suc$ we distinguish two cases:

Let $w \in L_f(q) \setminus L_m(q)$. Then there exists some $(a, \delta) \in \mathsf{A} \times \mathsf{L}$ such that $q_k \xrightarrow{(a,\delta)}_q$ but $t_k \not\xrightarrow{(a,\delta)}_t$. That is, $w \cdot (a, \delta) \in L_m(q)$ and so (by (3)) $w \cdot (a, \delta) \in L_m(p)$. Since p is purely nondeterministic, we can construct a finite computation $c' = (\langle q_{i-1}, t'_{i-1}\rangle(a_i, \delta_i)\langle q_i, t'_i\rangle)_{0<i\leq l} \in C(q,t)$ where $\text{proj}_t(c) = \text{proj}_t(c')$, $t'_l = t_k$, and $p_l \xrightarrow{(a,\delta)}_p$. The computation is maximal (since $t_k = t'_j \not\xrightarrow{(a,\delta)}_{t'}$) and unsuccessful (since $|c'| < \omega$ and $t'_l \notin Suc$). Therefore, p must$_{\mathbb{T}}$ t does not hold.

Let now $w \in L_m(q)$ (and thus $w \in L_m(p)$). We can then construct a maximal computation c' as above and then p must$_{\mathbb{T}}$ t does not hold given that q must$_{\mathbb{T}}$ t does not hold.

Finally, if $p \Downarrow w$ and $|c| = \omega$, since $\text{proj}_t(c) \notin \Pi_\omega(t)$, $\text{proj}_t(c) \in \Pi_\omega(q)$, and $w \in L_\omega(p)$, we can construct an infinite computation $c' \in C(q,t)$ such that $\text{proj}_t(c) = \text{proj}_t(c')$. Similar to the above, c' is unsuccessful and so p must$_{\mathbb{T}}$ t does not hold. All the cases lead to $p \sqsubseteq_{\mathbb{T}}^{must} q$, as desired. □

5 Conclusion and Future Work

We proposed in this paper a model of timed processes based on timed transition systems. We addressed the problem of infinite timed processes by developing a theory of timed ω-final states. This theory is new but inspired by the acceptance family of Müller automata [3]. We also extended the testing theory of De Nicola and Hennessy [9] to timed testing. We then studied the derived timed may and must preorders and developed alternative characterizations for them. These characterizations are very similar to the characterization of De Nicola and Hennessy's testing preorders, which shows that our preorders are fully back compatible: They extend the existing preorders as mentioned, but they do not take anything away.

The significance of our results stems from the fact that while algorithms and techniques for real-time testing have been studied actively [5, 13], the domain still lacks solid techniques and theories. Our paper

attempts to present a general theoretical framework for real-time testing, in order to facilitate the subsequent evolution of the area. To serve such a purpose our framework is as close as possible to the original framework of (untimed) testing, as shown in our characterization theorems. In addition, our characterization is surprisingly concise in terms of the test cases needed.

We also note that the algebraic and logic specifications attempt to achieve the same thing (conformance testing) in two different ways. Each of them is more convenient for certain systems, as they both have advantages and disadvantages: logic approaches allow loose specifications (and therefore greater latitude in their implementations) but lack compositionality, while algebraic specifications are compositional by definition but are often seen as too detailed (and therefore too constraining). Our immediate future work will focus on bringing logic and algebraic specifications together, thus obtaining heterogeneous specifications for real-time systems, that combine the advantages of the two paradigms. We will focus on providing a uniform basis for analyzing heterogeneous real-time system specifications with a mixture of timed transition systems and linear-time temporal logic (LTL) formulae. Our current work is to establish standard algorithms for constructing timed processes or timed transition system from LTL formulae (and the other way around).

References

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

[2] R. ALUR, C. COURCOUBETIS, AND D. DILL, *Model-checking for real-time systems*, in Proceedings of the 5th Annual Symposium on Logic in Computer Science, IEEE Computer Society Press, 1990, pp. 414–425.

[3] R. ALUR AND D. L. DILL, *A theory of timed automata*, Theoretical Computer Science, 126 (1994), pp. 183–235.

[4] A. BERTOLINO, *Software testing research: Achievements, challenges, dreams*, in Future of Software Engineering, IEEE, 2007, pp. 85–103.

[5] L. B. BRIONES AND E. BRINKSMA, *A test generation framework for quiescent real-time systems*, in Formal Approaches to Testing of Software, 2004, p. 7185.

[6] M. BROY, B. JONSSON, J.-P. KATOEN, M. LEUCKER, AND A. PRETSCHNER, eds., *Model-Based Testing of Reactive Systems: Advanced Lectures*, no. 3472 in Lecture Notes in Computer Science, Springer, 2005.

[7] S. D. BRUDA AND S. G. AKL, *Real-time computation: A formal definition and its applications*, International Journal of Computers and Applications, 25 (2003), pp. 247–257.

[8] A. COHN, *The notion of proof in hardware verification*, J. Autom. Reasoning, 5 (1989), pp. 127–139.

[9] R. DE NICOLA AND M. HENNESSY, *Testing equivalences for processes*, Theoretical Computer Science, 34 (1983), pp. 83–133.

[10] C. HOARE, *Communicating sequential processes*, Prentice Hall, 1985.

[11] ISO/IEC, *Information technology–Fibre Distributed Data Interface (FDDI)–Part 25: Abstract Test Suite for FDDI, Station Management Conformance Testing (SMT-ATS). International standard 9314-25*, 1998.

[12] L. LAMPORT, *Specifying Systems*, Addison-Wesley, 2002.

[13] J. SPRINGINTVELD, F. VAANDRAGER, AND P. R. D'ARGENIO, *Testing timed automata*, Theoretical Computer Science, 254 (2001), p. 225257.

[14] J. TRETMANS, *A formal approach to conformance testing*, in Protocol Test System, VI, Elsevier, 1994, pp. 257–276.

[15] ——, *Conformance testing with labelled transition systems: implementation relations and test generation*, Computer Networks and ISDN Systems, 29 (1996), pp. 49–79.