Timed Test Generation Based on Timed Temporal Logic

STEFAN D. BRUDA and CHUN DAI
Department of Computer Science
Bishop’s University
Sherbrooke, Quebec J1M 1Z7
CANADA
stefan@bruda.ca, cdai@cs.ubishops.ca

Abstract: Based on a previously developed testing theory for real-time systems, we show how timed tests can be algorithmically generated out of a timed variant of linear-time temporal logic (namely, TPTL), so that a process must pass the generated test if and only if the process satisfies the given temporal logic formula. Beside the obvious use of such an algorithm (to generate tests), our result also establishes a correspondence between timed must testing and timed temporal logic.

Key-words: Formal methods, Real-time systems, Model-based testing, May testing, Must testing, Testing preorders, Test generation, Timed temporal logic, TPTL

1 Introduction

How to guarantee validity and reliability of software and hardware is one of the most pressing problems. Conformance testing [14] is known in this area for its succinctness and high automatization. Its aim is to check whether an implementation conforms to a given specification.

System specifications can be mainly classified into two kinds: algebraic and logic. The first favours refinement, where a single algebraic formalism is equipped with a refinement relation to represent and relate both specifications and implementations [13]. An implementation is validated if it refines its specification. Process algebrae, labelled transition systems, and finite automata are commonly used in this classification, with traditional refinement relations being either behavioural equivalences or preorders [6, 10]. A typical example is model-based testing [6]. The second approach prefers assertive constructs. Different formalisms describe the properties of specifications and implementations; specifications are defined logically 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 [8].

The domain of conformance testing consists in reactive systems, which interact with their environment. 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. Real-time specifications are then used as the basis of conformance testing for such systems.

We previously developed [9] a semantic theory for real-time system specification. Using a theory of timed \( \omega \)-final states as well as a timed testing framework based on De Nicola and Hennessy’s testing [10], we developed timed may and must preorders that relate timed processes on the basis of their responses to timed tests. 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 abound, they mostly restrict the real-time domain to make it tractable. 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, so our theory is general. Still, it is more practical than one might expect. Indeed, the characterization of the timed preorders [9] uses a surprisingly concise set of timed tests.

To further address the practicality issue we tackle now automatic test generation. We show how to algorithmically build tests starting from timed propositional temporal logic [3] formulae. This algorithm is also a first (but significant) step toward an integration of operational and assertive specification styles in the area of real-time systems, to obtain heterogeneous (algebraic and logic) specifications and tools.

2 Preliminaries and Notations

Preorders are reflexive and transitive relations. \(|\mathbb{N}| = \omega\). Our constructions are based on some alpha-
The transition relation (we use timed traces are normal traces. One of them). To add time to a trace, we add time information of the events or states (but not the delays between them). To add time to a trace, we add time information of the events or states (but not the delays between them).

If $x$ is a clock and $r$ is a real then $x \sim r$ is a time constraint, $\sim \in \{\leq, <, =, \neq, >, \geq\}$. Constraints can be joined in conjunctions or disjunctions $T(C)$ is the set of time constraints over a set $C$ of clocks.

**Timed Transition Systems** Labelled transition systems (LTS) [6] serve as a semantic model for formal specification languages. A timed transition system is an LTS extended with time values associated to actions. Timed automata [2] introduce the notion of time constraints. We find convenient to combine the two concepts into a unified model for real time, which we call by abuse of terminology timed transition system.

For a set $A$ of observable actions ($\delta \notin A$), a set $L$ of times values, and a set $C$ of clocks, a timed transition system is a tuple $(\langle A \times L \rangle \cup \{\tau\}, C, S, \rightarrow, p_0)$, where: $S$ is a countable set of states; every state $p \in S$ has an associated clock interpretation $c_p : C \rightarrow L$; $\rightarrow \subseteq (S \times (A \times L) \times S \times T(C) \times 2^C) \cup (S \times \{\tau\} \times S)$ is the transition relation (we use $p_{\tau_{\rightarrow}}$ instead of $(p, (a, \delta), p', \tau, C) \in \rightarrow$, omitting $C$ whenever $C = \emptyset$ and also $\tau$ whenever $\tau = \tau$); $p_0$ is the initial state.

Whenever $p_{\tau_{\rightarrow}}$, the transition system performs an action with delay $\delta$; the delay causes the clocks to progress so that $c_p(x) = c_p(x) + \delta$ whenever $x \notin C$ and $c_p(x) = 0$ otherwise; the transition is enabled only if $\tau$ holds under the interpretation $c_p$; $\tau$ transitions do not affect clock interpretations and cannot be time constrained. Normally a trace is described as a sequence of the events or states (but not the delays between them). To add delay to a trace, we add delay information to the usual notion of trace (that contains actions only). A timed trace over $A$, $L$, and $C$ is a member of $(A \times L \times T(C) \times 2^C)^* \cup (A \times L \times T(C) \times 2^C)^\omega$.

If both $L$ and $T(C)$ (or equivalently $C$) are empty the timed transition system becomes an LTS, and its timed traces are normal traces. One of $L$ or $T(C)$ could be empty and we still obtain a timed trace; we use this to differentiate processes and specifications.

As usual $p_{\tau_{\rightarrow}}$ if $p \rightarrow \cdots \rightarrow p_{n_{\tau_{\rightarrow}}} p$ and $p \Rightarrow p'$ if $p = p_0 \rightarrow \cdots \rightarrow p_n = p'$, for some $n \geq 0$, and $p \xrightarrow{\omega} q$ whenever $w = (a_i, \delta_i, t_i, C_i)_{0 \leq i \leq k}$ and $p_{\tau_{\rightarrow}} p_{n_{\tau_{\rightarrow}}} p_{n_{\tau_{\rightarrow}}} p_{n_{\tau_{\rightarrow}}} p_{n_{\tau_{\rightarrow}}} p_k = p'$.

A timed path $\pi$ of a timed transition system $M = ((A \times L) \cup \{\tau\}, C, S, \rightarrow, p_0)$ is a potentially infinite sequence $(p_{i-1}, (a_i, \delta_i, t_i, C_i), p_i)_{0 \leq i \leq k}$, where $p_{i-1} \tau_{\rightarrow} p_i$ for all $0 \leq i \leq k$; If $|\pi| = \omega$ (with $|\pi| = k$), $\pi$ is infinite; otherwise $\pi$ is finite. If $|\pi| \in \mathbb{N}$ and $p_{\tau_{\rightarrow}}$ (i.e., $p_{\tau_{\rightarrow}}$ is a deadlock state), then the timed path $\pi$ is called maximal. trace($\pi$), the (timed) trace of $\pi$ is defined as the sequence $(a_i, \delta_i, t_i, C_i)_{0 \leq i \leq |\pi|, a_i \not\in \tau} \in (A \times L \times T(C) \times 2^C)^\ast$. If $f(p')$, $f_m(p')$, $f_l(p')$, $f_{D}(p')$, $f_{\omega}(p')$ denote the sets of all finite timed paths, all maximal timed paths, and all infinite timed paths starting from state $p' \in S$, respectively. We also put $f_{\omega}(p') = f_{\omega}(p') \cup f_m(p') \cup f_l(p')$.

State $p'$ of transition system $p$ is timed divergent, denoted by $p' \uparrow_p$ (or just $p' \uparrow$ when there is no ambiguity), if $\exists}\beta \in f_{\omega}(p') : \text{trace}(\beta) = \varepsilon$. State $p'$ is called timed $\omega$-divergent (denoted by $p' \uparrow_{\omega}$) for some $w = (a_i, \delta_i, t_i, C_i)_{0 \leq i \leq \omega} \in (A \times L \times T(C) \times 2^C)^\ast \cup (A \times L \times T(C) \times 2^C)^\omega$, if $\exists p \in S, k \leq k, p' \xrightarrow{w} p'' \uparrow_p$, with $w' = (a_i, \delta_i, t_i, C_i)_{0 \leq i \leq \omega}$. State $p'$ is timed convergent or timed $\omega$-convergent ($p' \downarrow_p$ and $p' \downarrow_p w$, respectively) if it is not the case that $p' \uparrow_p$ and $p' \downarrow_p w$, respectively. The set of initial actions of $p'$ is init$_p(p') = \{(a, \delta, t, C) \in A \times L \times T(C) \times 2^C : \exists p' = p'(a, \delta, t, C)\}$.

For a timed transition system (state) $p$ the timed finite-trace language $L_f(p)$, maximal-trace (complete-trace) language $L_m(p)$, infinite-trace language $L_l(p)$, and divergence language $L_{D}(p)$ of $p$ are

$$L_f(p) = \{\text{trace}(\pi) : \pi \in f_{\omega}(p)\}$$
$$L_m(p) = \{\text{trace}(\pi) : \pi \in f_m(p)\}$$
$$L_l(p) = \{\text{trace}(\pi) : \pi \in f_l(p)\}$$
$$L_{D}(p) = \{w \in (A \times L \times T(C) \times 2^C)\ast \cup (A \times L \times T(C) \times 2^C)^\omega : p \uparrow_p w\}$$

We defined timed languages slightly differently from the original [2] to reflect their use for system specification and also to simplify the presentation. However if we omit the clocks and their constraints (which we will do for processes) there is a natural bijection between our definition and the original.

Once more similar to the theory of timed automata [2] we introduce a set of timed $\omega$-final states. Then the timed $\omega$-regular trace language of some timed transition system $p$ is $L_{\omega}(p) = \{\text{trace}(\pi) : \pi \in \omega\}$.
Ilω(p) ⊆ (A × L × T(C) × 2C)ω, where Ilω(p) contains exactly all the ω-regular timed paths; i.e., ω-final states must occur infinitely often in any π ∈ Ilω(p). We also exclude henceforth Zeno behaviours from all the languages that we consider: no trace is allowed to show Zeno behaviour. In other words, time progresses and must eventually grow past any constant value.

Timed Propositional Temporal Logic Timed Propositional Temporal Logic (TPTL) [3] is one of the most general temporal logics with time constraints [4]. TPTL extends linear-time temporal logic (LTL) [4, 8] by adding time constraints, so that its semantics is given with respect to timed traces1, i.e., timed words in (A × L)* ⊆ (A × L)ω. We use TPTL without congruence, but we just call it TPTL for short.

With φ, ψ, ρ ranging over TPTL formulae, τ ranging over A, x ranging over a set of clocks C, and c ranging over positive constants, the syntax of the term θ and the TPTL formula φ is the following:

\[
\begin{align*}
\theta &= x + c \mid c \\
\phi &= \theta_1 \leq \theta_2 \mid \top \mid \bot \mid a \mid \neg \phi \mid \phi_1 \wedge \phi_2 \\
&\quad \wedge X \phi \mid \phi_1 \parallel \phi_2 \mid \tau \phi \mid i_0 \leq i < k
\end{align*}
\]

F is the set of all TPTL formulae. A timed trace \( w = (a_i, \delta_i)_{0 \leq i \leq k} \subseteq (A × L)* ⊆ (A × L)ω \) satisfies φ iff \( w \models_\gamma \phi \). The relation \( \models_\gamma \subseteq ((A × L)*) ⊆ (A × L)ω × F \) is the least relation satisfying the conditions in the semantics of TPTL formulae shown below, with \( w_j \) standing for \( (a_i, \delta_i)_{j \leq i \leq k} \) for any \( 1 \leq j \leq k \), and \( \gamma : C \rightarrow L \) being some clock interpretation.

- \( \theta_1 \leq \theta_2 \) iff \( \gamma(\theta_1) \leq \gamma(\theta_2) \).
- \( w \models_\gamma \top \) and \( w \models_\gamma \bot \) for any \( w \).
- \( w \models_\gamma a \) iff \( w \neq \varepsilon \) and \( a_1 = a \).
- \( w \models_\gamma \neg \phi \) iff \( w \models_\gamma \phi \).
- \( w \models_\gamma \phi_1 \wedge \phi_2 \) iff \( w \models_\gamma \phi_1 \) and \( w \models_\gamma \phi_2 \).
- \( w \models_\gamma X \phi \) iff \( w \models_\gamma \phi \).
- \( w \models_\gamma \phi \) \( \parallel \phi_2 \) iff \( \exists \bar{i} < \bar{i} \leq k : \forall i \leq r < k : w_r \models_\gamma \phi_2 , \forall 0 < s < t : w_s \models_\gamma \phi_1 \).
- \( w \models_\gamma x.\phi \) iff \( w \models_\gamma (\phi_{/x}) \).

We require that \( \gamma(x + c) = \gamma(x) + c \) and \( \gamma(c) = c \); \( \gamma[t/x] \) is the clock interpretation that agrees with \( \gamma \) on all clocks except \( x \), which is mapped to \( t \in L \).

As usual one can also introduce the derived operators G (“globally”) and F (“eventually”) as \( G \phi = \bot R \phi \) and \( F \phi = \top U \phi \), respectively. The operator R (“releases”) is the dual of the operator U. A timed process \( p \) satisfies the TPTL formula \( \phi \), written \( p \models \phi \), iff \( \forall w \in L_f(p) \cup L_m(p) \cup L_\omega(p) \cup L_D(p) : w \models_\gamma \phi \).

3 Previous Work

We adapted [9] the theory of De Nicola and Hennessy [10] to timed testing. Then we introduced [9] the concept of Büchi success for tests, so that the properties of infinite runs of a process can be readily identified by tests. We used timed transition systems as the formalism for both processes and tests.

Definition 1 [9]. A timed process \(((A × L) \cup \{\tau\}, S, \succeq, p_0)\) is a timed transition system \(((A × L) \cup \{\tau\}, \emptyset, S, \succeq, p_0)\) with an empty set of clocks (and thus with no time constraints). It follows that all the traces of any timed process are in the set \((A × L)* \cup (A × L)ω\).

A timed test \((A \cup \{\tau\}, C, T, \rightarrow_i, \Sigma, \Omega, t_0)\) is a timed transition system \(((A × L) \cup \{\tau\}, C, T, \rightarrow_i, t_0)\) with the addition of \( \Sigma \subseteq T \) of success states and \( \Omega \subseteq T \) of ω-final states. Note that \( L = \emptyset \) for tests and therefore \( \rightarrow_i \subseteq (T × A × T(C) × 2^C × T) \cup (T × \{\tau\} × T) \).

The transition relation of a process and a test are restricted (in different manners) because the test runs in parallel with the process under the test1. This latter process (called the implementation) features time sequences but no time constraints, while the test features only time constraints. The set of all timed tests is denoted by \( T \).

Definition 2 [9]. A partial computation \( c \) of a timed process \( p \) and a timed test \( t \) is a potentially infinite sequence \( (p_{i-1}, t_{i-1})_{(a_i, \delta_i)}_{i=0}^{i=k} \) of states, where \( k \in \mathbb{N} \cup \{\omega\} \), such that \( p_i \in P \) and \( t_i \in T \) for all \( 0 < i \leq k \); and \( \delta_i \in L \) is taken from \( p \), \( t_i \), and \( C_i \) are taken from \( t \), and \( R \in \{1, 2, 3\} \) for all \( 0 < i \leq k \). The relation \( \hookrightarrow \) is defined by the following rules:

- \( \langle p_{i-1}, t_{i-1} \rangle \hookrightarrow (p_i, t_i) \) if \( a_i = \tau \) and \( p_{i-1} \xrightarrow{\tau} p_i \), \( t_{i-1} = t_i \), and \( t_i \notin \Sigma \).
- \( \langle p_{i-1}, t_{i-1} \rangle \hookrightarrow (p_i, t_i) \) if \( a_i = \tau \) and \( p_{i-1} = p_i \), \( t_{i-1} \xrightarrow{\tau} t_i \), and \( t_i \notin \Sigma \).

1A timed process is a timed transition system without time constraints, as detailed in Section 3.

2Note however that the difference is syntactical only, for indeed the transition relation for a timed process allows for an empty set L.
The projection \( \proj^c_p(c) \) of \( c \) on \( p \) is defined as:

\[
\{ p_{i-1}, (a_i, \delta_i), p_i \}_{i \in \C_i} \in \Pi(p), \text{ where } I^c_p = \{ 0 < i \leq k : R_i \in \{1, 3\} \}.
\]

Similarly, the projection \( \proj^t(c) \) of \( c \) on \( t \) if defined as:

\[
\{ t_{i-1}, (a_i, \delta_i), t_i \}_{i \in \C_i} \in \Pi(t), \text{ where } I^c_t = \{ 0 < i \leq k : R_i \in \{2, 3\} \}.
\]

**Definition 3 [9].** A partial computation \( c \) is a computation whenever:

- \( k \neq t \)
- \( \forall t \in T \) if \( p^c_k \neq p^c_i \)
- \( \forall t \in T \) if \( p^c_k = p^c_i \)

**Definition 4 [9].** \( p \) may pass \( t \) (\( p \) may \( T \)), iff there exist at least one successful computation \( c \in C(p, t) \):

\[
\forall t \in T : p^c_k \implies q^c_m \implies q^c_t \text{ and } p^c_m \implies q^c_t.
\]

**Proposition 1 [9]**

1. \( p \equiv^T q \) iff \( L_f(p) \subseteq L_f(q) \) and \( L_w(p) \subseteq L_w(q) \).
2. \( p \equiv^T q \) iff for all \( w \in (A \times L)^* \), if \( |w| \leq \omega \) it holds that:

   - \( a \in A \) and \( \forall t \), \( a \equiv^T q \).
   - \( p \equiv^T q \) and \( \forall t \), \( p \equiv^T q \).
   - \( p \equiv^T q \) and \( \forall t \), \( p \equiv^T q \).

The second characterization is given in terms of timed trace inclusions, more similar to the characterization of other preorders [1, 7].

A timed process \( p \) is purely nondeterministic, if

\[
\forall \text{states } p', p^T \equiv^T p \text{ implies } p' \equiv^T p.
\]

**Proposition 2 [9]**

Let \( p \) and \( q \) be timed processes such that \( p \) is purely nondeterministic. Then \( p \equiv^T q \) iff all of the following hold:

1. \( L_D(q) \subseteq L_D(p) \)
2. \( L_f(q) \setminus L_D(q) \subseteq L_f(p) \)
3. \( L_m(q) \setminus L_D(q) \subseteq L_m(p) \)
4. \( L_w(q) \setminus L_D(q) \subseteq L_w(p) \)

**Theorem 3**

Given a TPTL formula \( \phi \) there exists a test \( T_\phi \) such that \( p \equiv^T \phi \) if and only if for any arbitrary \( \gamma \), \( p \equiv^T \phi \) if \( p \equiv^T T_\phi \) for any timed process \( p \). \( T_\phi \) can be algorithmically constructed starting from \( \phi \).

**Proof.** Let \( p \) be an arbitrary timed process. \( T_\phi \) is first constructed to consider only infinite computations, and will then be modified to consider maximal traces. A sub-formula of \( \phi \) is defined inductively:

1. \( \phi \) is a sub-formula of \( \phi \).
2. Any formula \( t \) formed by terms of form \( \theta \) and relational and boolean operators (henceforth called "time formula") occurring in \( \phi \) is a sub-formula of \( \phi \), but no sub-formula of \( t \) is a sub-formula of \( \phi \) (a time formula is indivisibly a sub-formula),
3. If \( \neg \xi \) is a sub-formula of \( \phi \), then so is \( \xi \),
4. If \( \Omega \) is a sub-formula then so is \( \xi, \Omega \in \{X, x\} \).
5. If \( \xi_1 \Omega \xi_2 \) is a sub-formula of \( \phi \) then so are \( \xi_1 \) and \( \xi_2, \Omega \in \{\lor, \land, \lor\} \).

**Construction of \( T_\phi \)**

Let \( C_\phi \) be the set of exactly all the clocks that occur in a TPTL formula \( \phi \), and let \( C(\phi) \) be the closure of \( \phi \), i.e., the set of exactly all the sub-formulas of \( \phi \). Furthermore, let \( \Theta(\phi) \subseteq C(\phi) \) contain exactly all the sub-formulas of \( \phi \) that are time formulas.

The construction of \( T_\phi \) is then based on the untimed construction developed by Vardi and Wolper [15]. We first consider the "local" automaton \( L_\phi = (2\phi, C_\phi, \Pi, \tau, \emptyset, (0, \Pi, s)) \).
1. $\psi \in s$ iff $\neg \psi \notin s$ for all $\psi \in C(\phi)$.
2. $\xi \wedge \psi \in s$ iff $\xi \in s$, $\psi \in s$ for all $\xi \wedge \psi \in C(\phi)$.
3. $x.\psi \in s$ implies $\psi \in s$.

The transition relation is defined as $s \xrightarrow{a \in C} t$ iff $a = t$, $C = \{ x \in C_\phi : x.\psi \in s \wedge \psi \in t \}$, $t = \lambda(\Theta(\phi) \cap s)$, $\psi \in t \wedge x.\psi \in s$ implies $x.\psi \notin t$, and

1. $s = s_0$ and $\phi \in a$, or
2. $s \neq s_0$, for all $\psi \in C(\phi)$, $X \psi \in s$ iff $\psi \in t$, and for all $\xi \cup \phi \in C(\phi)$ either $\psi \in s$, or $\xi \in s$ and $\xi \cup \psi \in t$.

$L_\phi$ does not impose any acceptance conditions as mentioned, but enforces all the time constraints present in the original formula $\phi$. Indeed, at every moment frozen in time by a construction $x.\psi$ we reset the respective clock in the local automaton (for the set $C$ of clocks reset by a transition out of $s$ contains exactly all the sets of clocks $x$ reset by an $x$. construction in $s$). Later, whenever a time formula is encountered, that formula is added to the time constraints that enable the transition. Checking the time formula to determine that the transition is enabled has the intended effect: the time formula needs to be true for the whole formula $\phi$ to be true, and the semantics of time in a timed transition system ensures that every clock measures the time from when it was reset in the transition system (i.e., frozen in the formula) to the current time.

Acceptance is handled by the “eventuality” automaton $E_\phi = (2C(\phi), \emptyset, 2C(\phi), \tau E, \emptyset, \emptyset)$, with $E(\phi) = \{ \xi \cup \psi \in C(\phi) \}$ and $s \xrightarrow{a \in C} t$ iff

1. $s = \emptyset$ and $\xi \cup \psi \in t$ iff $\psi \notin a$ for all $\xi \cup \psi \in a$,
2. $s \neq \emptyset$ and $\xi \cup \psi \in t$ iff $\psi \notin a$ for all $\xi \cup \psi \in s$.

The eventuality automaton is identical to the one developed elsewhere [15]. It tries to satisfy the eventualities of the formula (with no regard for time constraints). The current state keeps track of which eventualities have yet to be satisfied.

The test $T_\phi$ is obtained by taking the cross-product of $L_\phi$ and $E_\phi$. The cross-product is taken using the usual (untimed) construction [15], for only the transitions in $\tau L$ contain time constraints and/or clock resets (and these go into the composite automaton together with the actions that accompany them in $L_\phi$). This test characterizes traces over $2C(\phi) \times L$; in order to switch to $A \times L$ we project over $A$ the action labels of all the transitions, as done previously [15].

The construction of $T_\phi$ follows carefully the construction for the untimed case. It is then immediate that $T_\phi$ is correct as far as untimed words are concerned, in the sense that $\text{trace}(\text{proj}_c(c)) \models_\gamma \phi$ for exactly all the successful infinite computations $c \in C(p, T_\phi)$ stripped of time information. The timing information is added (via $L_\phi$), as detailed above. In all,

$$\text{trace}(\text{proj}_c(c)) \models_\gamma \phi$$

for exactly all the successful infinite computations $c \in C(p, T_\phi)$ (1)

We now enhance $T_\phi$ so that it also accepts finite maximal traces: For every state $s$ in $T_\phi$, we check if all the formulae contained in $s$ are satisfied by the trace $\varepsilon$. Checking for acceptance of the trace $\varepsilon$ (like for any fixed trace) can be done algorithmically along the structure of the formula $\phi$. Then, for every state $s$ in $T_\phi$ such that each TPTL formula $\phi$ labeling $s$ is satisfied by $\varepsilon$, we add a transition $s \xrightarrow{\phi} \Delta$, where $\Delta$ is a new state. We use $\Delta$ to distinguish from other states also having no outgoing transitions; these states represent deadlocks due to inconsistent sub-formulae of $\phi$. The final states of $T_\phi$ will then be the set containing only the state $\Delta$ thus introduced. We have:

$$\text{trace}(\text{proj}_c(c)) \models_\gamma \phi$$

for exactly all the successful maximal computations $c \in C(p, T_\phi)$ (2)

Indeed, $\chi(p) \models_\gamma \phi$ (with $\chi(x) = \text{trace}(\text{proj}_x(c))$) implies $s_0 \xrightarrow{\chi(T_\phi)} s \xrightarrow{\phi} \Delta$ (where $s_0$ is the initial state of $T_\phi$). That is $s_0 \xrightarrow{\chi(T_\phi)} s \xrightarrow{\phi}$. Thus, $c$ is maximal. Conversely, if $c$ is successful and maximal, then there exits $s_0 \xrightarrow{\chi(T_\phi)} s \xrightarrow{\phi} T_\phi$. According to the algorithm there exits then $s_0 \xrightarrow{\chi(T_\phi)} s \xrightarrow{\phi} \Delta$. Thus, $c$ is maximal.

That $p \models_\gamma \phi$ iff $p$ must $T_\phi$ follows from Properties (1) and (2), as desired.

5 Conclusion

Our previous work [9] proposed a model of timed tests. We addressed the problem of characterizing infinite behaviour of timed processes, then we extended the testing theory of De Nicola and Hennessy to timed testing. We then studied the derived timed may and must preorders and developed alternative characterizations.

We now presented an algorithm for test generation out of TPTL formulae. Both processes and tests are represented by timed transition systems instead of automata (i.e., their number of states is not necessarily finite). This is consistent with the huge body of similar constructs in the untimed domain. The timed
tests produced out of TPTL formulae (Theorem 3) are always finite automata (meaning that their set of states is always finite). This is quite nice to have for a very practical reason, as infinite-state tests must be further refined to become practical characterization tools.

One significance of our results stems from the fact that while algorithms and techniques for real-time testing have been studied actively [5, 12], the domain still lacks solid techniques and theories. Our work (previous and also in this paper) attempts to present a general theoretical framework for real-time testing 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 and also in the test generation algorithm. Our characterization is also surprisingly concise in terms of the test cases needed [9].

Beside the obvious use for test generation, our algorithm also relates the satisfaction relation of TPTL to the must operator. We note that the algebraic and logic specification techniques 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. Our test generation algorithm also forms the basis of bringing logic and algebraic specifications together, thus obtaining heterogeneous specifications for real-time systems that combine the advantages of the two paradigms. This has the potential of providing a uniform basis for analyzing heterogeneous real-time system specifications with a mixture of timed transition systems and timed logic formulae.

We consider TPTL without congruence because the theory of timed transition systems does not offer a congruence mechanism. Such a mechanism could likely be introduced without much difficulty, but to our knowledge none of the temporal logics used in practical settings take congruence into consideration, so we preferred to leave time transition systems intact and exclude congruence from TPTL instead.

We (continue to) avoid the discussion of discrete versus continuous time. All the results and definitions are oblivious to whether time is considered discrete or continuous. We therefore leave the decision of discreteness to the future uses of this work.

References: