A Testing Framework for Real-Time Specifications

by

Chun Dai

A thesis submitted to the

Department of Computer Science

in conformity with the requirements for

the degree of Master of Science

Bishop's University

Sherbrooke, Quebec, Canada

September 2008

Copyright © Chun Dai, 2008

Abstract

This thesis proposes a new semantic model for reasoning about real-time system specifications featuring a combination of timed processes and formulas in linear-timed temporal logic with time constraints (TLTL). Based on a theory of timed omega-final states, the thesis presents a framework of timed testing and two refinement timed preorders similar to De Nicola and Hennessy's may and must testing. The paper also provides 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. The thesis then establishes a tight connection between TLTL formula satisfaction and the timed must-preorder. More precisely, it is shown that a timed labeled transition system satisfies a TLTL formula if and only if it refines an appropriately defined timed process constructed from the formula. Consequently, we developed a timed must-preorder which allows for a uniform treatment of traditional notions of process refinement and model checking under time constraints. The implications of this novel theory are illustrated by means of a simple example system, in which some components are specified as transition systems and others as TLTL formulas.
Key words: real time, real-time transition system, timed process, timed preorder, timed testing, conformance testing, model checking, LTL.

Acknowledgments

This paper cost many people's time and energy. My deepest gratitude goes first and foremost to my supervisor, Professor Stefan D. Bruda, who led me into the world of conformance testing. I sincerely thank him for his constant encouragement and guidance. He has helped me a lot through all the stages of the writing of this paper and provided to me many useful reference books. Without his constant instruction, this paper could not have reached this present form.

I also owe my thanks to my dear friends and classmates, who have done me a favor in helping me work out my problems during the difficult course of the paper.

I would like to deeply thank the all various people who, during the several months in which this endeavor lasted, provided me with useful and helpful assistance. Without their care and consideration, this thesis would likely not have been finished.

1 Introduction 1

1.1 Thesis motivation 1

1.2 Thesis contribution 3

1.3 Thesis organization.. 4

2 Preliminaries... 5

2.1 Timed automata and timed transition systems 5

3 Timed Processes, Timed Traces and Timed Languages.. 8

3.1 Timed ω-languages and ω-final states..11

4 A Testing Theory 14

4.1 Timed tests and timed testing preorders 14

4.2 Alternative characterizations 18

5 Timed must-testing and linear-time temporal logic. 31

5.1 Syntax and Semantics of TLTL. 31

5.2 Constructing Timed Processes from TLTL Formulas.. 35

5.2.1 Constructing Timed Process 35

5.2.2 Allowing Finite Maximal Timed Traces. 45

5.2.3 Allowing ω-final Timed Traces 47

5.2.4 Allowing Divergent Timed Traces... 48

5.2.5 Allowing all Timed Traces.. 50

5.3 Relating TLTL Satisfaction to the Timed Must-preorder... 52

6 Parallel composition 54

7 Motivating Example 58

8 Conclusions 62

9 Open problems 65

References 66

1 Introduction

1.1 Thesis motivation

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 [1,2].Among many theoretical methods for this, conformance testing [3,4] 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 [5], together with implementations, can be classified mainly 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 [6,7]. An implementation is validated correct if it refines its specification. Since it often defines the system transitionally, process algebrae [8], labelled transition systems [7], and finite automata [9] are commonly used, with traditional refinement relations being either behavioural equivalences or preorders [7, 10]. A typical example is model-based testing [7]. The second approach to conformance testing prefers assertive constructs; different formalisms are used to describe the properties of the system specifications and implementations [7,11]. 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 [11].

The domain of conformance testing includes 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 sequence of events, they must satisfy constraints on the 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 [5] 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 heterogeneous system specifications featuring mixtures of real-time transition systems and formulas in linear-time temporal logic with time constraints (TLTL). 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 [10] as starting points, 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, the syntax and semantics of TLTL. We set up and refine an algorithm for constructing timed processes from TLTL formulas together with a parallel composition operator for interface of different parts in the specification so that we can apply our testing on the heterogenous

specifications. Our testing framework is as close to the original framework of (untimed) testing as possible, and is also as general as possible. While many 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 immediately applicable 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.

1.2 Thesis contribution

The key result of this paper is that TLTL model checking can be reduced to model-based testing. More precisely, a timed process T_ϕ can be constructed from a TLTL formula ϕ in such a way that a timed labelled transition system satisfies ϕ if and only if it is larger than T_ϕ with respect to the timed must-preorder. Whenever a heterogeneous system is presented, the specification needs to be converted into a unified form in order to solve the problem of conformance checking. One way of doing this is by converting TLTL formulas into timed processes, which can then be verified using algebraic methods. The other way is to express everything using TLTL formulas, which can then be verified using logical methods. Since we have chosen timed preorders as the starting point to set up the framework of timed testing, we prefer constructing

timed processes to TLTL formulas. We also show that our must-preorder is compositional for a parallel composition operator, and then illustrate our technical results by a small example featuring the heterogeneous design of an airline boarding system.

1.3 Thesis organization

The thesis is organized as follows: In the next chapter, some preliminaries are presented. In the third chapter, we introduce our notion of timed processes. Chapter 4 defines the framework of timed testing and timed testing preorders. In Chapter 5, we define the syntax and the semantics of the temporal logic that is used in the thesis and then the connection between timed must-testing and TLTL model checking is investigated. A parallel composition operator is developed in Chapter 6 while Chapter 7 applies our specification framework to a simple example. The last two chapters contain our conclusions and open problems.

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. We denote $|\mathbb{N}|$ by ω.

2.1 Timed automata and timed transition systems

Our theory is based on an *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. ΦC is the set of time constraints over a set C of clocks. A *clock interpretation* for a set C is a mapping C $\mapsto$ 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 $x \in C$. 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 ~ 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 $(\mathrm{A} \times \mathrm{L})^* \cup (\mathrm{A} \times \mathrm{L})^\omega$.

The basic idea of *labelled transition system* (LTS) [6] is to fix an alphabet (which usually need not be finite) of labels and to define the evaluation relation → as a relation on triples of an expression, a label, and another expression.

Formally, a labelled transition system is a 4-tuple $(S,\ L, \rightarrow, s_0)$, where S is a set of states, L is a set of labels, $\rightarrow \subseteq \mathrm{S} \times (\mathrm{L} \cup \{\tau\}) \times \mathrm{S}$ is a the transition relation, and s_0 is the initial state.

Timed automata [12] 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 [12]: 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 $E \subseteq S \times S \times \Sigma \times 2^C \times \Phi C$ is the transition relation. A member $(s, s', a, \lambda, \phi)$ of the transition relation represents a transition from 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 latter 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 $(S, (\mathrm{A} \times \mathrm{L}) \cup \{\tau\}, \rightarrow, s_0)$, where S is a countable, non-empty set of states; A is a set of visible actions with $\tau \notin \mathrm{A}$ representing the special, internal action; L is a set of time values (which we informally call time actions); $\rightarrow \subseteq S \times (\mathrm{A} \times \mathrm{L}) \cup \{\tau\} \times S$ 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 1. *For a set* A *of observable actions (*$\tau \notin \mathrm{A}$*), a set* L *of times values, and a set* C *of clocks with an associated set* $\Phi\mathrm{C}$ *of time constraints, a timed process is a tuple* $((\mathrm{A} \times \mathrm{L}) \cup \{\tau\}, C, S, \rightarrow, (s_0, c_0))$*, where*

– S is a countable set of states; p = (s, c) $\in$ *S, where* s *is the location (or label) of the state and* c *is a clock interpretation over* C[1]*;*

$- \rightarrow \subseteq S \times ((\mathrm{A} \times \mathrm{L}) \cup \{\tau\}) \times S \times \Phi C$ *is the transition relation. Commonly, we use* $p \underset{\Phi c}{\overset{(a,\delta)}{\rightarrow}} p'$ *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 $(s, c) \underset{\Phi c}{\overset{(a,\delta)}{\rightarrow}} (s', c')$, the process performs a with delay δ; the delay causes the clocks to progress so that $c' = c + \delta$; the transition is enabled

---

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

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

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 sequence of 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 values only).

Definition 2. *A timed trace over A, L, and* $\Phi\mathrm{C}$ *is a member of* $(\mathrm{A}\times\mathrm{L}\times\Phi\mathrm{C})^{*}$ $\cup(\mathrm{A}\times\mathrm{L}\times\Phi\mathrm{C})^{\omega}$, *where A is a finite set of events, L is a set of time actions and* $\Phi\mathrm{C}$ *is a set of time constraints.*

If both L and $\Phi\mathrm{C}$ 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 $\Phi\mathrm{C}$ could be empty and we still obtain a timed trace; this will be used later.

We will use the following relation: $p \overset{(a,\delta)}{\underset{\Phi C}{\Rightarrow}} p'$ iff $p = p_0 \overset{\tau}{\rightarrow} p_1 \overset{\tau}{\rightarrow} p_2 \overset{\tau}{\rightarrow} \ldots \overset{\tau}{\rightarrow} p_n \overset{(a,\delta)}{\underset{\Phi C}{\rightarrow}} p'$ for some $n \geq 0$, and $p \overset{\varepsilon}{\Rightarrow} p'$ iff $p = p_0 \overset{\tau}{\rightarrow} p_1 \overset{\tau}{\rightarrow} p_2 \overset{\tau}{\rightarrow} \ldots \overset{\tau}{\rightarrow} p_n = p'$ for some $n \geq 0$. By abuse of notation, we also write $p \overset{w}{\Rightarrow} q$ whenever $w = (a_i, \delta_i, \Phi c_i)_{0<i\leq k}$ and $p \overset{(a_1,\delta_1)}{\underset{\Phi c_1}{\Rightarrow}} p_1 \overset{(a_2,\delta_2)}{\underset{\Phi c_2}{\Rightarrow}} p_2 \overset{(a_3,\delta_3)}{\underset{\Phi c_3}{\Rightarrow}} p_3 \ldots \overset{(a_k,\delta_k)}{\underset{\Phi c_k}{\Rightarrow}} p_k = p'$.

Definition 3. *Let* $M = ((\mathrm{A}\times\mathrm{L})\cup\{\tau\}, C, S, \rightarrow, (s_0, c_0))$ *be a timed process. A timed path* $\pi(M)$ *is a potentially infinite sequence* $(<(s_{i-1}, c_{i-1}), (a_i, \delta_i, \Phi c_i),$

$(s_i, c_i) >)_{0<i\leq k}$, *where* $(s_{i-1}, c_{i-1}) \underset{\Phi c}{\overset{(a_i,\delta_i)}{\Rightarrow}} (s_i, c_i)$, *for all* $0 < i \leq k$ *with* $a_i \in \mathrm{A}, \delta_i \in \mathrm{L}$, $\Phi c_i \in \Phi C$

We use |π| to refer to k, the length of π. If |π| = ω, 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 $(a_i, \delta_i, \Phi c_i)_{0\leq i\leq|\pi|} \in (\mathrm{A}\times\mathrm{L}\times\Phi C)^* \cup (\mathrm{A}\times\mathrm{L}\times\Phi C)^\omega$.

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 $(s',c') \in S(\text{or}\, p \in S)$, respectively. We also put $\Pi(p') = \Pi_f(p') \cup \Pi_m(p') \cup \Pi_I(p')$. The empty timed path π (with |π| = 0) is symbolized by () and its (always empty) trace by *ε*.

We can now introduce the different languages associated with a timed process p.

Definition 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*

$$L_f((s,c)) = \{trace(\pi) \mid \pi \in \prod_f((s,c))\} \quad \subseteq (\mathrm{A}\times\mathrm{L}\times\Phi C)^*$$

$$L_m((s,c)) = \{trace(\pi) \mid \pi \in \prod_m((s,c))\} \quad \subseteq (\mathrm{A}\times\mathrm{L}\times\Phi C)^*$$

$L_I((s,c)) = \{trace(\pi) \mid \pi \in \prod_I((s,c))\} \quad \subseteq (\mathrm{A} \times \mathrm{L} \times \Phi C)^* \cup (\mathrm{A} \times \mathrm{L} \times \Phi C)^\omega$

Then the set of initial actions of state *p' = (s', c')* in process p is defined as follows: $\mathrm{I}_p((s',c')) = \{(a,\delta,\Phi c) \in (\mathrm{A} \times \mathrm{L} \times \Phi \mathrm{C}) : \exists (s'',c'') \cdot (s',c') \overset{(a,\delta)}{\underset{\Phi c}{\Rightarrow}}{}_p (s'',c'')\}$.

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 [12]. A timed word over an alphabet *Σ* consists in a pair (*σ*, *t*), where $\sigma = \sigma_1\sigma_2...$ 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 isomorphism between our definition and the original [12]), in order to reflect the use of such languages for system specifications (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 2.1) $v = v_1 \cdot v_2 \cdot v_3 \ldots$. 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) · · · , whose infinite time-event sequence is $(a,1)^{\omega}$. Another time trace might be (a, 1)(a, 1/2)(a, 1/4) · · · , whose infinite time-event sequence is (a, 1)(a, 1/2)(a, 1/4) · · · . If L has no positive lower-bound on the time length, then timed traces over L may exhibit Zeno behaviours, as in 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 [12, 13]).

In all, we define the timed ω-regular-trace language as follows.

Definition 5. *The timed* ω*-regular-trace language of some process p is* $L_{\omega}(p) = \{trace(\pi) : \pi \in \Pi_{\omega}(p)\} \subseteq (\mathrm{A} \times \mathrm{L} \times \Phi\mathrm{C})^{*} \cup (\mathrm{A} \times \mathrm{L} \times \Phi\mathrm{C})^{\omega}$, *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 *divergent*, denoted by $(s',c')\Uparrow_p$, if $\exists\pi \in \Pi_I(s',c'), trace(\pi)=\varepsilon$. State ($s'$, c') is called (time) w-divergent (denoted by $(s',c')\Uparrow_p w$) for some $w=(a_i,\delta_i,\Phi C_i)_{0<i<k} \in (\mathrm{A}\times\mathrm{L}\times\Phi\mathrm{C})^* \cup (\mathrm{A}\times\mathrm{L}\times\Phi\mathrm{C})^\omega$ if one can reach a divergent state starting from (s', c') when executing a finite prefix of w, i.e., if $\exists l \in \mathbb{N}, (s'',c'') \in S : l \le k, (s',c') \overset{w'}{\Rightarrow} (s'',c'')\Uparrow_p$, with $w'=(a_i,\delta_i,$ $\Phi c)_{0<i\le l}$. For convenience we write $L_D(p')$ for the divergence language of p', i.e., $L_D(p')=\{w \in (\mathrm{A}\times\mathrm{L}\times\Phi\mathrm{C})^* \cup (\mathrm{A}\times\mathrm{L}\times\Phi\mathrm{C})^\omega \mid p'\Uparrow_p w\}$.

Conversely, state (s', c') is (time) convergent or (time) 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 chapter we extend the testing theory of De Nicola and Hennessy [10] to timed testing. The traditional testing framework defines behavioural preorders that relate labelled transition systems according to their responses to tests [14]. Tests are used to verify the external interactions between a system and its environment. We use 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 6. *A* timed test $((\mathrm{A} \times \mathrm{L}) \cup \{\tau\}, C, T, \rightarrow_t, \Omega, (s_0^t, c_0^t), Suc)$ *is a timed*

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

process $((\mathrm{A}\times\mathrm{L})\cup\{\tau\},\mathrm{C},T,\rightarrow,(s_0,c_0))$ *with the addition of a set* $Suc\in T$ *of* success states *and a set* $\Omega\in T$ *of* ω-*final states. Furthermore,* $\mathrm{L}=\varnothing$ *for tests and therefore* $\rightarrow_t\subseteq S\times(\mathrm{A}\cup\{\tau\})\times S\times\Phi\mathrm{C}$.

The transition relation differs from the original one because the test runs in parallel with the process under test[4]. This latter process (called the implement-tation) features time sequences but no time constraints, while the test features only time constraints. It is meaningless to run the test by itself. If $\Phi\mathrm{C}=\varnothing$ 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 7. *A* partial computation c *with respect to a timed process* p *and a timed test* t *is a potentially infinite sequence* $(<p_{i-1},t_{i-1})>\underset{\Phi c_i}{\overset{(a_i,\delta_i)}{\longmapsto}}_R<p_i,t_i>)_{0<i\leq k}$, *where* $k\in\mathbb{N}\cup\{\omega\}$, *such that* (1) $p_i\in P$ *and* $t_i\in T$ *for all* $0<i\leq k$, *and* (2) $(a_i,\delta_i)\in(\mathrm{A}\times\mathrm{L})\cup\{\tau\}$ *is taken from p,* Φc_i *is the time constraint (if any) taken from* t *and* $\mathrm{R}\in\{1,2,3\}$ *for all* $0<i\leq k$.

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

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

$$-< p_{i-1},t_{i-1} > \overset{(a_i,\delta_i)}{\underset{\Phi c_i}{\mapsto_1}} < p_i,t_i > if\ \ \alpha_i = \tau,\ p_{i-1} \overset{\tau}{\underset{p}{\rightarrow}} p_i, t_{i-1} = t_i,\ and\ \ t_{i-1} \notin Suc$$

$$-< p_{i-1},t_{i-1} > \overset{(a_i,\delta_i)}{\underset{\Phi c_i}{\mapsto_2}} < p_i,t_i > if\ \ \alpha_i = \tau,\ p_{i-1} = p_i,\ t_{i-1} \overset{\tau}{\underset{t}{\rightarrow}} t_i, and\ \ t_{i-1} \notin Suc$$

$$-< p_{i-1},t_{i-1} > \overset{(a_i,\delta_i)}{\underset{\Phi c_i}{\mapsto_3}} < p_i,t_i > if\ \ (\alpha_i,\delta_i) \in (A \times L),\ p_{i-1} \overset{(a_i,\delta_i)}{\rightarrow_p} p_i, t_{i-1} \overset{(a_i,\delta_i)}{\underset{\Phi c_i}{\rightarrow_t}} t_i, and\ \ t_{i-1} \notin Suc$$

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 $proj_p(c)$ of c on p is defined as $(< p_{i-1},((\alpha_i,\delta_i)), p_i >)_{i \in I_P^C}$ $\in \prod(p)$, where $I_P^C = \{0 < i \leq k \mid R_i \in \{1,3\}\}$. Similarly, the projection $proj_t(c)$ of c on t if defined as $(< t_{i-1},((\alpha_i,\delta_i),\Phi c_i),t_i >)_{i \in I_t^C} \in \prod(t)$, where $I_t^C = \{0 < i \leq k \mid R_i \in \{2,3\}\}$.

Definition 8. *A partial computation c is called computation if it satisfies the following properties:* (1) *c is maximal, i.e.,* $k \in \mathbb{N}$ *implies* $p_k \overset{\tau}{\not\rightarrow}_p, t_k \overset{\tau}{\not\rightarrow}_t$ *and*

$I_p(p_k) \cap I_t(t_k) = \varnothing$ *(* 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 =* ω *implies* $proj_p(c) \in \prod_I(p)$.

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

Definition 9. *Computation c is called successful if* $t_{|c|} \in Suc$ *whenever* $|c| \in \mathbb{N}$*, and* $proj_p(c) \in \prod_I(t)$ *whenever |c| =*ω.

Definition 10. p may pass t, *denoted by* $p\ may_T\ t$*, if there exists at least one successful computation* $c \in C(p,t)$*. Analogously,* p must pass t, *denoted by* $p\ must_T\ t$ *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 classical testing theory [10]—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 11. *Let p and q be timed processes. Then,*

– $p \sqsubseteq_T^{may} q$ *iff* $\forall t \in \Gamma,\ p\ may_T\ t \quad \Rightarrow \quad q\ may_T\ t$

– $p \sqsubseteq_T^{must} q$ *iff* $\forall t \in \Gamma,\ p\ must_T\ t \quad \Rightarrow \quad q\ must_T\ t$

It is immediate that the relations $\sqsubseteq_T^{may}$ and $\sqsubseteq_T^{must}$ 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 [14] and relates timed testing directly with the behaviour of processes.

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

2 $p \sqsubseteq_T^{must} q$ *iff for all* $w \in ((\mathrm{A} \times \mathrm{L})^* \cup (\mathrm{A} \times \mathrm{L})^\omega)$ *such that* $p \Downarrow w$ *it holds that:*

(a) $q \Downarrow w$

(b) if $|w| < \omega$ *then* $\forall q', q \overset{w}{\Rightarrow} q'$ *implies* $\exists p', p \overset{w}{\Rightarrow} p'$ *and* $I_p(p') \subseteq I_q(q')$

(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 [6]. Note that we are now concerned with $\sqsubseteq_T^{must}$ only, as the simpler $\sqsubseteq_T^{must}$ is already characterized in terms of timed traces in Theorem 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) $p' \overset{\tau}{\rightarrow}_p$ implies $p' \overset{a}{\not\rightarrow}_p$ and $|\{((a,\delta),p''): p' \overset{(a,\delta)}{\rightarrow}_p p''\}| = 1$. Note that every timed process p can be transformed into 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 $p' \overset{(a,\delta)}{\rightarrow}_p p''$ into two transitions $p' \overset{\tau}{\rightarrow}_p p_{<p',(a,\delta),p''>}$ and $p_{<p',(a,\delta),p''>} \overset{(a,\delta)}{\rightarrow}_p p''$, where $p_{<p',(a,\delta),p''>}$ is a new, distinct state.

Theorem 2. *Let p and q be timed processes such that p is purely nondeterministic. Then,* $p \sqsubseteq_T^{must} q$ *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. $L_m(q) \setminus L_D(q) \subseteq L_m(p)$, *and*
4. $L_\omega(q) \setminus L_D(q) \subseteq L_\omega(p)$.

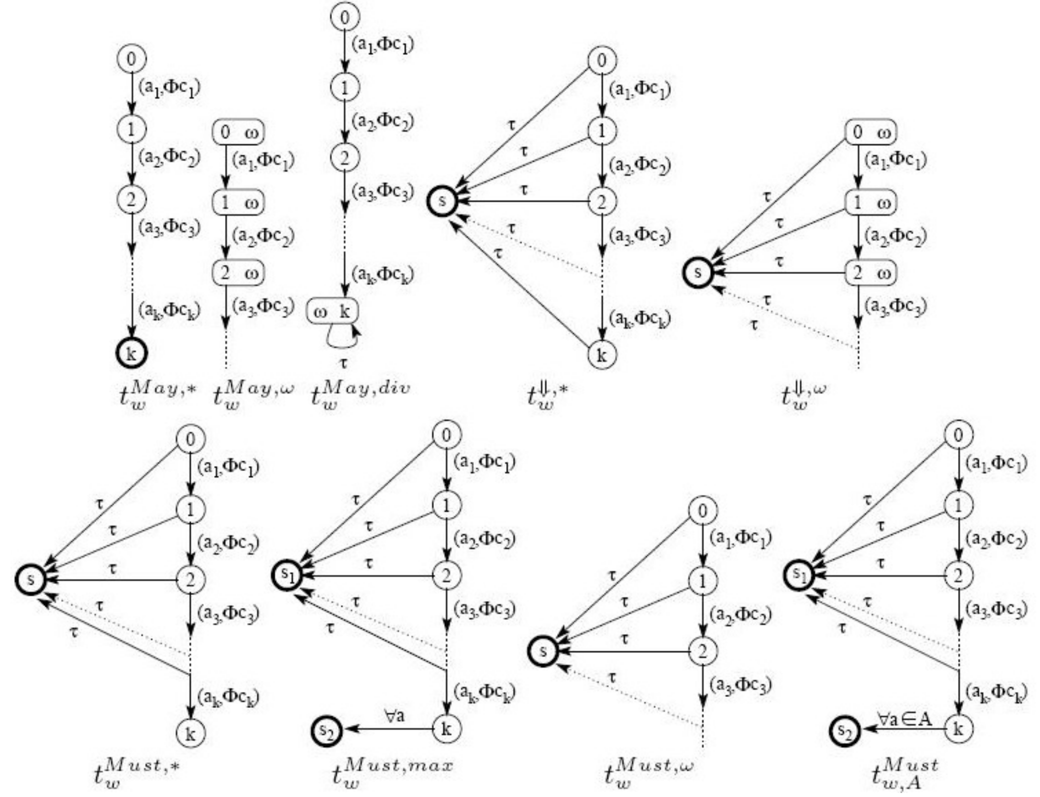

Fig. 1. Timed tests used for the characterization of timed may and must preorders.

With respect to finite traces, the characterizations of timed tests differ from the ones of classical preorders by the addition of time variables. Here, we do not limit the number of clocks, but since every action between two states requires at most one time constraint, the number of clocks is associated with the number of time constraints. 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 proof of the two characterization theorems rely on the properties of the following specific timed tests.

– For $w=(a_i,\delta_i)_{0<i\le k}\in(\mathrm{A}\times\mathrm{L})^*$, let $t_w^{May,*} =< A\cup\{\tau\},\mathrm{C},T,\rightarrow,\phi,0,k)$, where

$T = \{0,1,...,k\}$, and $\rightarrow = \{< i-1, (a_i, i, c_i = \Sigma^i_{j=0}\delta_i) : 0 < i \le k\}$.

- For $w = (a_i, \delta_i)_{0<i\le k} \in (\mathrm{A}\times\mathrm{L})^\omega$, let $t_w^{May,\omega} =< \mathrm{A}\cup\{\tau\}, \mathrm{C}, T, \rightarrow, T, 0, \phi)$, where $T = \mathbb{N}, \rightarrow = \{(i-1, (a_i, i, c_i = \Sigma^i_{j=0}\delta_i) : i > 0\}$.
- For $w = (a_i, \delta_i)_{0<i\le k} \in (\mathrm{A}\times\mathrm{L})^*$, let $t_w^{May,div} =< \mathrm{A}\cup\{\tau\}, \mathrm{C}, T, \rightarrow, \{k\}, 0, \phi)$, where $T = \{0,1,...,k\}, \rightarrow = \{(i-1, a_i, i, c_i = \Sigma^i_{j=0}\delta_i) : 0 < i \le k\} \cup \{k, \tau, k, \mathrm{True}\}$.
- For $w = (a_i, \delta_i)_{0<i\le k} \in (\mathrm{A}\times\mathrm{L})^*$, let $t_w^{\Downarrow,*} = (\mathrm{A}\cup\{\tau\}, \mathrm{C}, T, \rightarrow, \phi, 0, \{s\})$, where $T = \{0,1,...,k\} \cup \{s\}$, $\rightarrow = \{(i-1, a_i, i, c_i = \Sigma^i_{j=0}\delta_i) : 0 < i \le k\} \cup \{(i, \tau, s, \mathrm{True}) : 0 < i \le k\}$
- For $w = (a_i, \delta_i)_{0<i\le k} \in (\mathrm{A}\times\mathrm{L})^*$, let $t_w^{\Downarrow,\omega} =< \mathrm{A}\cup\{\tau\}, \mathrm{C}, T, \rightarrow, \{s\}, 0, \{s\})$, where $T = \mathbb{N} \cup \{s\}, \rightarrow = \{(i-1, a_i, i, c_i = \Sigma^i_{j=0}\delta_i) : i > 0\} \cup \{(i, \tau, s, \mathrm{True}) : i > 0\}$
- For $w = (a_i, \delta_i)_{0<i\le k} \in (\mathrm{A}\times\mathrm{L})^*$, let $t_w^{Must,*} = (\mathrm{A}\cup\{\tau\}, \mathrm{C}, T, \rightarrow, \phi, 0, \{s\})$, where $T = \{0,1,...,k\} \cup \{s\}, \rightarrow = \{(i-1, a_i, i, c_i = \Sigma^i_{j=0}\delta_i) : 0 < i \le k\} \cup \{(i, \tau, s, \mathrm{True}) : 0 \le i < k\}$
- For $w = (a_i, \delta_i)_{0<i\le k} \in (\mathrm{A}\times\mathrm{L})^*$, let $t_w^{Must,\max} = (\mathrm{A}\cup\{\tau\}, \mathrm{C}, T, \rightarrow, \phi, 0, \{s_1, s_2\})$, where $T = \{0,1,...,k\} \cup \{s_1, s_2\}, \rightarrow = \{(i-1, a_i, i, c_i = \Sigma^i_{j=0}\delta_i) : 0 < i \le k\} \cup \{(i, \tau, s_1, \mathrm{True}) : 0 \le i < k\} \cup \{(k, a, s_2, \mathrm{True}) : (a, \Phi c) \in \mathrm{A}\times\Phi\mathrm{C}\}$
- For $w = (a_i, \delta_i)_{0<i\le k} \in (\mathrm{A}\times\mathrm{L})^\omega$, let $t_w^{Must,\omega} = (\mathrm{A}\cup\{\tau\}, \mathrm{C}, T, \rightarrow, \phi, 0, \{s\})$, where $T = \mathbb{N} \cup \{s\}, \rightarrow = \{(i-1, a_i, i, c_i = \Sigma^i_{j=0}\delta_i) : i > 0\} \cup \{(i, \tau, s, \mathrm{True}) : i \in \mathbb{N}\}$.
- For $w = ((a_i, \delta_i)_{0<i\le k} \in (\mathrm{A}\times\mathrm{L})^*$ and $A \subseteq \mathrm{A}$, let $t^{Must}_{w,(A,L)} =< \mathrm{A}\cup\{\tau\}, \mathrm{C}, T, \rightarrow, \phi, 0, \{s_1, s_2\})$, where $T = \{0,1,...,k\} \cup \{s_1, s_2\}, \rightarrow = \{< i-1, (a_i, i, c_i = \Sigma^i_{j=0}\delta_i) : 0 < i \le k\} \cup \{(i, \tau, s_1, \mathrm{True}) : 0 \le i < k\} \cup \{(k, a, s_2, True) : a \in A\}$

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 the timed tests $t_w^{May,*}$ and $t_w^{May,\omega}$ test for the presence of a finite and infinite trace w, respectively, the 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. The 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, the timed test $t_{w,(A,L)}^{Must}$ is capable of comparing the initial action sets of states reached when executing trace w with respect to a subset $A \subseteq \mathrm{A}$. Note that we use the tightest time constraint possible in our test. We denote $c_i = \Sigma_{j=0}^{i} \delta_i$ by Φc_i in what follows (and also in Figure 1).

Our specific timed tests satisfy the following desired properties:

Lemma 1.

1. *Let* $w \in (\mathrm{A} \times \mathrm{L})^*$. *Then,* $w \in L_f(p)$ *iff* $p\ may_T\ t_w^{May,*}$
2. *Let* $w \in (\mathrm{A} \times \mathrm{L})^\omega$. *Then,* $w \in L_\omega(p)$ *iff* $p\ may_T\ t_w^{May,\infty}$
3. *Let* $w \in (\mathrm{A} \times \mathrm{L})^*$. *Then,* $w \in L_\omega(p)$ *iff* $p\ may_T\ t_w^{May,div}$
4. *Let* $w \in (\mathrm{A} \times \mathrm{L})^*$. *Then,* $p \Downarrow w$ *iff* $p\ must_T\ t_w^{\Downarrow,*}$
5. *Let* $w \in (\mathrm{A} \times \mathrm{L})^* \cup (A, L)^\omega$. *Then,* $p \Downarrow \mathrm{w}$ *iff* $p\ must_T\ t_w^{\Downarrow,\omega}$
6. *Let* $w \in (\mathrm{A} \times \mathrm{L})^*$ *such that,* $p \Downarrow w$. *Then,* $w \notin L_f(p)$ *iff* $p\ must_T\ t_w^{Must,*}$
7. *Let* $w \in (\mathrm{A} \times \mathrm{L})^*$. *such that,* $p \Downarrow w$. *Then,* $w \notin L_m(p)$ *iff* $p\ must_T\ t_w^{Must,\max}$
8. *Let* $w \in (\mathrm{A} \times \mathrm{L})^\omega$ *such that,* $p \Downarrow w$. *Then,* $w \notin L_\omega(p)$ *iff* $p\ must_T\ t_w^{Must,\omega}$

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 $w = (a_i, \delta_i)_{0<i\le k}$ for some $k \in \mathbb{N} \cup \{\omega\}$.

– Item 1, $\Rightarrow$: $w \in L_f(p)$, and thus $p_0 \overset{(a_1,\delta_1)}{\Rightarrow} p_1 \overset{(a_2,\delta_2)}{\Rightarrow} \cdots \overset{(a_k,\delta_k)}{\Rightarrow} p_k$ (Definition 4). On the other hand, $t_0^{May,*} \underset{\Phi c_1}{\overset{(a_1,\delta_1)}{\Rightarrow}} t_1^{May,*} \underset{\Phi c_2}{\overset{(a_2,\delta_2)}{\Rightarrow}} \cdots \underset{\Phi c_k}{\overset{(a_k,\delta_k)}{\Rightarrow}} t_k^{May,*}$ (definition of $t_w^{May,*}$ including the form of Φc_i). Therefore, $(< p_{i-1}, t_{i-1}^{May,*} >) \underset{\Phi c_i}{\overset{(a_i,\delta_i)}{\mapsto}} (< p_i, t_i^{May,*} >)_{0<i\le k}$, 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, $t_k^{May,*} \overset{\tau}{\nrightarrow}$ and $I_p(p_k) \cap I_t(t_k^{May,*}) = \varnothing$), and is further the trace of a successful computation (since $t_k^{May,*} \in Suc$). It then follows that $p\ may_T\ t_w^{May,*}$.

$\Leftarrow$: Given that $p\ may_T\ t_w^{May,*}$, we have a successful computation c of p and $t_w^{May,*}$. That is, $(<p_{i-1}, t_{i-1}^{May,*}>) \underset{\Phi c_i}{\overset{(a_i,\delta_i)}{\mapsto}} (<p_i, t_i^{May,*}>)_{0<i\leq k}$, $t_k^{May,*} \overset{\tau}{\nrightarrow}, I_p(p_k) \cap I_t($ $t_k^{May,*}) = \phi)$, and $t_w^{May,*} = t_k^{May,*} \in Suc$. By a reverse argument we conclude then that $w \in L_f(p)$ $(t_0^{May,*} \underset{\Phi c_1}{\overset{(a_1,\delta_1)}{\Rightarrow}} t_1^{May,*} \underset{\Phi c_2}{\overset{(a_2,\delta_2)}{\Rightarrow}} \cdots \underset{\Phi c_k}{\overset{(a_k,\delta_k)}{\Rightarrow}} t_k^{May,*}$, then $p_0 \overset{(a_1,\delta_1)}{\Rightarrow} p_1 \overset{(a_2,\delta_2)}{\Rightarrow} \cdots \overset{(a_k,\delta_k)}{\Rightarrow} p_k$, and thus $w \in L_f(p)$).

- Items 2 and 3 are proven similarly.

- Item 4, $\Rightarrow$: Assume that $p\ must_T\ t_w^{\Downarrow,*}$ 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 $p_0 \overset{(a_1,\delta_1)}{\Rightarrow} p_1 \overset{(a_2,\delta_2)}{\Rightarrow} \cdots \overset{(a_l,\delta_l)}{\Rightarrow} p_l \overset{\tau}{\rightarrow}$ $p_l \cdots$ which means that $p \Uparrow w$, a contradiction. So it must be that $p\ must_T\ t_w^{\Downarrow,*}$.

  $\Leftarrow$: Assume that $p \Uparrow w$. Then for some $0 < l \leq k$ there exists one trace $p_0 \overset{(a_1,\delta_1)}{\Rightarrow} p_1 \overset{(a_2,\delta_2)}{\Rightarrow} \cdots \overset{(a_l,\delta_l)}{\Rightarrow} p_l \overset{\tau}{\rightarrow} p_l \cdots$ which fails the test $t_w^{\Downarrow,*}$. This contradicts the condition $p\ must_T\ t_w^{\Downarrow,*}$ and so it must be that $p \Downarrow w$.

- Item 5 is proven similarly.

– Item 6, $\Rightarrow$: Assume that $p\ must_T\ t_w^{Must,*}$ does not hold. According the definition of $t_w^{Must,*}$, there are two ways for p to fail the test: Either $p_0 \stackrel{(a_1,\delta_1)}{\Rightarrow} p_1 \stackrel{(a_2,\delta_2)}{\Rightarrow} \cdots \stackrel{(a_l,\delta_l)}{\Rightarrow} p_l \stackrel{\tau}{\rightarrow} p_l \cdots$, or $p_0 \stackrel{(a_1,\delta_1)}{\Rightarrow} p_1 \stackrel{(a_2,\delta_2)}{\Rightarrow} \cdots \stackrel{(a_k,\delta_k)}{\Rightarrow} p_k$. 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 $p\ must_T\ t_w^{Must,*}$.

– Item 7 and 8 are proven similarly. □

The proof of Theorem 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 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 $p\ may_T\ t_w^{May,*}$. Since $p \sqsubseteq_T^{May} q$ it follows that p $may_T\ t_w^{May,*}$ and thus $w \in L_f(p)$.

$w \in L_\omega(p)$ has two sub-cases:

1. If $|w| = \omega$, then $p\ may_T\ t_w^{May,\omega}$. Since $p \sqsubseteq_T^{May} q$ it follows that $p\ may_T\ t_w^{May,\omega}$ and thus $w \in L_\omega(p)$.
2. If $|w| < \omega$, then $p\ may_T\ t_w^{May,div}$. Since $p \sqsubseteq_T^{May} q$ it follows that p $may_T\ t_w^{May,div}$ and thus $w \in L_\omega(p)$.

We go now to the $\Leftarrow$ direction for Item 1. Let t be any timed process such that $p\ may_T\ t$, i.e., there exists a successful computation $c \in C(p,t)$ with $w = trace$ $(proj_p(c)) = trace(proj_t(c))$.

1. If $|w| = \omega$, when $w \in L_\omega(p)$ 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 $w = trace(proj_q(c')) = trace(proj_t(c'))$ and $proj_t(c') = proj_t(c)$. It follows that $q\ may_T\ t$ and therefore $p \sqsubseteq_T^{May} q$.
2. 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 $q\ may_T\ t$ as above.

On to Item 2 now. For the $\Rightarrow$ direction we have that $p \sqsubseteq_T^{Must} q$, $w \in (\mathrm{A} \times \mathrm{L})^* \cup$ $(\mathrm{A} \times \mathrm{L})^\omega$ such that $p \Downarrow w$. Then,

1. $p\ must_T\ t_w^{\Downarrow,*}$ or $p\ must_T\ t_w^{\Downarrow,\omega}$ (Lemma 1), then $q\ must_T\ t_w^{\Downarrow,*}$ or $q\ must_T\ t_w^{\Downarrow,\omega}$ (Definition 11), thus $q \Downarrow w$ (Lemma 1).

2. It could be that $|w| = \omega$ or not, so we distinguish two cases.

   (a) $|w| < \omega$: Let $q \overset{w}{\Rightarrow} q'$ for some q', i.e., $w \in L_f(q)$. Assume that there is no

p' such that $p \overset{w}{\Rightarrow} p'$ and $I_p(p') \subseteq I_q(q')$.

i. Suppose that $p \overset{w}{\not\Rightarrow}$ i.e., $w \notin L_f(p)$. Then $p\ must_T\ t_w^{Must,*}$ does not hold (contrapositive of Lemma 1), a contradiction.

ii. Suppose now that $p \overset{w}{\Rightarrow}$. Let then $X = \{(a,\delta) \in I_p(p') : p \overset{w}{\Rightarrow} p'\} \neq \phi$. 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 that $p\ must_T\ t_{w,B}^{Must}$ (by the construction of $t_{w,B}^{Must}$); however, it is not the case that $q\ must_T\ t_{w,B}^{Must}$ (since $q' \overset{(a,\delta)}{\not\Rightarrow}$ for any $(a,\delta) \in (B,L)$). This contradicts the assumption that $p \sqsubseteq_T^{must} q$.

(b) $|w| = \omega$: Assume that $w \notin L_\omega(p)$. Then $p\ must_T\ t_w^{Must,\omega}$ (Definition 11) and thus $w \notin L_\omega(q)$ (Lemma 1). This contradicts $w \in L_\omega(q)$ (given).

Finally, for the $\Leftarrow$ direction of Item 2, let t be any timed process such that $q\ must_T\ t$ does not hold, i.e., there exists an unsuccessful computation $c = (< q_{i-1}, t_{i-1} >, (a_i, \delta_i, \Phi c_i), < q_i, t_i >)_{0<i\leq k} \in C(q,t)$ (Definition 10). Let $w = trace(proj_p(c)) = trace(proj_t(c))$.

1. Assume that $p \Downarrow w$. 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 $proj_p(c') \in \Pi_\omega(p)$ and $proj_t(c') \notin \Pi_\omega(t)$ (because $|proj_p(c')| < \omega$). This implies that $p\ must_T\ t$ does not hold (Definition 10) and thus $p \sqsubseteq_T^{must} q$ (since q $must_T\ t$ does not hold by the contrapositive of Definition 11)

2. Assume now that $p \Downarrow w$, i.e., $w \notin L_D(p)$. Then,

   (a) If $|c| < \omega$ then: (*i*) $w \in L_f(q)$, $q \overset{w}{\Rightarrow} q'$ for some *q'* and $t_k \neq Suc$ by definition of $t_w^{Must,*}$, (*ii*) $q_k \overset{\tau}{\not\to}, t_k \overset{\tau}{\not\to}, I_q^c(q_k) \cap I_t^c(t_k) = \phi$ by definition of $t_w^{Must,\max}$; and (*iii*) $\exists p' : p \overset{w}{\Rightarrow} p', I_p^c(p') \subseteq I_q^c(q')$ by condition.

   (b) By observations (*i*)-(*iii*) we have a finite computation $c' = (< p_{i-1}, t'_{i-1} >,$ $(a_i, \delta_i, \Phi c_i), < p_i, t'_i >_{0<i\leq l} \in C(p,t)$ with $proj_t(c') = proj_t(c)$ and $< p_l,$ $t_l > = < p'', t_k >$, where $p' \overset{\varepsilon}{\Rightarrow} p''$ for some $p'' \overset{\tau}{\not\to}_p$. 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 (*i*) and (*ii*) 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 $p\ must_T\ t$ does not hold.

3. If $|c| = \omega$ then $q\ must_T\ t_w^{Must,\omega}$ does not hold. It follows that $w \in L_\omega(q)$, and thus $w \in L_\omega(p)$ (given). So $p\ must_T\ t_w^{Must,\omega}$ does not hold either (contra-positive of Lemma 1). In all, $p \sqsubseteq_T^{must} q$, as desired. □

The proof of Theorem 2 also relies on the properties of the timed tests introduced in Lemma 1.

Proof of Theorem 2 For the $\Rightarrow$ direction, assume that $p \sqsubseteq_T^{must} q$ and let $w \in (\mathrm{A} \times \mathrm{L})^* \cup (\mathrm{A} \times \mathrm{L})^\omega$. Then,

(1) $w \in L_D(q)$ implies $q \Uparrow w$, so it is not the case that $q\ must_T\ t_w^{\Downarrow,\omega}$

(Lemma 1(5)). Therefore it is not the case that $p\ must_T\ t_w^{\Downarrow,\omega}$ (since $p \sqsubseteq_T^{must} q$), 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 1(4)). In addition, it is not the case that $q\ must_T\ t_w^{Must,*}$ (Lemma 1(6)) and thus $p\ must_T\ t_w^{Must,*}$ does not hold (since $p \sqsubseteq_T^{must} q$). 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 1(7) and Lemma 1(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 $q\ must_T\ t$ does not hold (if such a test does not exist then $p \sqsubseteq_T^{must} q$ for any process *p*). Thus there exists an unsuccessful computation $c = (< q_{i-1}, t_{i-1} > (a_i, \delta_i) < q_i, t_i >)_{0<i\leq k} \in C(q,t)$, with $w = trace(proj_q(c)) = trace(proj_t(c))$.

1. 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 $p\ must_T\ t$.
2. If $p \Uparrow w$, $|c| < \omega$, and $t_k \notin Suc$:

   (a) Let $w \in L_f(q) \setminus L_m(q)$. Then there exists some $(a,\delta) \in \mathrm{A} \times \mathrm{L}$ such that

$q_k \xrightarrow{(a,\delta)}_q$ but $t_k \stackrel{(a,\delta)}{\nrightarrow}_t$. 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 $c' = (< q_{i-1}, t'_{i-1} > (a_i, \delta_i) < q_i, t'_i >)_{0<i\leq l} \in C(q,t)$ where $proj_t(c) = proj_t(c')$, $t'_l = t_k$, and $p_l \xrightarrow{(a,\delta)}_p$. The computation is maximal (since $t_k = t'_j \xrightarrow{(a,\delta)}_{t'}$) and unsuccessful (since $|c'| < \omega$ and $t'_l \notin Suc$). Therefore, $p\ must_T\ t$ does not hold.

(b) Let $w \in L_m(q)$ (and thus $w \in L_m(p)$). We can then construct a maximal computation *c'* as above and then $p\ must_T\ t$ does not hold given that $q\ must_T\ t$ does not hold.

3. If $p \Downarrow w$ and $|c| = \omega$, since $proj_t(c) \notin \Pi_\omega(t)$, $proj_t(c) \notin \Pi_\omega(q)$, and $w \in L_\omega(p)$, we can construct an infinite computation $c' \in C(q,t)$ such that $proj_t(c) = proj_t(c')$. Similar to the above, *c'* is unsuccessful and so $p\ must_T\ t$ does not hold.

All the cases lead to $p \sqsubseteq_T^{must} q$, as desired. □

5 Timed must-testing and linear-time temporal logic

In this chapter, we establish a connection between the timed must-preorder $\sqsubseteq_T^{must}$ and the satisfaction relation $\models$ for linear-time temporal logic with time constraints (TLTL). More specifically, our goal is to show how to construct a timed process P_ϕ from a TLTL formula ϕ in such a way that $q \models \phi$ if and only if $P_\phi \sqsubseteq_T^{must} q$, for any timed process q. Our result builds on timed Muller automata [12] approaches to LTL model checking [10,15,16,17,18,19].

5.1 Syntax and Semantics of TLTL

Our variant of TLTL interprets formulas with respect to sequences of actions with time constraints, and states. Accordingly, atomic propositions will also be interpreted with respect to actions and their time delays. Moreover, our variant extends traditional LTL [15,16,20], in that its semantics is given with respect to infinite and finite timed traces, i.e., timed words in $(\mathrm{A} \times \mathrm{L})^* \cup (\mathrm{A} \times \mathrm{L})^\omega$. This allows formulas to constrain ongoing as well as deadlocking behavior [16] for both actions and time.

The formal syntax of TLTL formulas is defined by the following BNF.

$\phi ::=$ true | false | (a,ψ) | $\neg(a,\psi)$ | $\phi \wedge \phi$ | $\phi \vee \phi$ | $\mathrm{X}_\Delta \phi$ | $\hat{\mathrm{X}}_\Delta \phi$ | $\phi U_\Delta \phi$ | $\phi R_\Delta \phi \mid \phi \rightarrow_\Delta \phi \mid \phi \rightarrow\!|_\Delta \phi$

Here, $(a,\psi) \in (A,\Psi)$ is an atomic proposition that is true for action (a,δ), whose δ is within the time constraint ψ, and false for any other action within the time constraint ψ or for no action in the time constraint. ψ consists in two real numbers x and y, the lower and upper bound, respectively. An example of time constraint is $[1,4)$. $\hat{\mathrm{X}}_\Delta \phi$ is the dual of the next-state operator $\mathrm{X}_\Delta \phi$. A typical way to add a metric for time is to allow the definition of bounded operators.[20] Inspired by this, Δ, is the time constraint for some operators, which is the same form as ψ but limits the time between two formulas with an operator. It does not only apply on the modal operators such as X, U, R, but also on the truth-functional operators such as ∧ ,∨,→.

Definition 12 *The time constraints* ψ and Δ are formed by two time bounds, lower time bound x and upper time bound y. A time constraint is an interval of time values, which may or may not be closed at any end. Examples include: $[x,y]$, $[x,y)$, $(x,y]$ or (x,y) with $x \in R^+, y \in R^+ \cup +\infty, y \geq x$.[5]

If an action (a,δ) occurs within the time bounds ψ, we say that δ

[5] When $y = +\infty$, the interval cannot be closed on the y side, so we only have $[x,+\infty)$ or $(x,+\infty)$

satisfies ψ, written $\delta \models \psi$. Also if the time interval $[\delta, \delta']$ is included in the time interval Δ, we say that $[\delta, \delta']$ satisfies Δ, written $[\delta, \delta'] \models \Delta$.

In the following, we denote the set of all TLTL formulas by F. Recall the definition of timed traces in Chapter 3. We say that a timed trace $w = (a_i, \delta_i)_{0<i\leq k} \in (\mathrm{A}\times\mathrm{L})^* \cup (\mathrm{A}\times\mathrm{L})^\omega$ satisfies ϕ iff $w \models \phi$ holds ($k = |w|$ is the length of w). The relation $\models \in ((\mathrm{A}\times\mathrm{L})^* \cup (\mathrm{A}\times\mathrm{L})^\omega) \times \mathrm{F}$ is the least relation satisfying the conditions in the semantics of TLTL formulas shown below, with w_j standing for $(a_i, \delta_i)_{j\leq i\leq k} \in (\mathrm{A}\times\mathrm{L})^*$, for any $1 \leq j \leq k$. We also say that a timed process p satisfies the TLTL formula ϕ, written $p \models \phi$, if $\forall w \in L_{\mathrm{m}}(p) \cup L_\omega(p) \cup L_D(p)$, $w \models \phi$. It should be noted that our syntax limits the application of negation to actions and time, rather than generally defining a formula $\neg\phi$ with meaning $w \models \neg\phi$ iff $w \not\models \phi$. This is not a restriction since our logic is self-dual. The operators $\wedge$ and $\vee$, X_Δ and $\hat{\mathrm{X}}_\Delta$, U_Δ and R_Δ, $\phi \rightarrow_\Delta \phi$ and $\phi \rightarrow\!|_\Delta\, \phi$ are dual to each other.

The semantics of TLTL:

$w \models true$ and $w \not\models false$

$w \models (a, \psi)$ if $w \neq \varepsilon$ and $a_i = a$ when $\delta_i \models \psi$

$w \models \neg(a, \psi)$ if $w \not\models (a, \psi)$

$w \models \phi_1 \wedge \phi_2$ if $w \models \phi_1$ and $w \models \phi_2$

$w \models \phi_1 \vee \phi_2$ if $w \models \phi_1$ or $w \models \phi_2$

$w \models X\phi$ if $w \neq \varepsilon$ and $w' \models \phi$ (with $w = (\alpha, \delta)w'$)

$w \models \hat{X}\phi$ if $w \neq \varepsilon$ implies $w' \models \phi$

$w \models \phi_1 \rightarrow_\Delta \phi_2$ if $\exists 0 < j < i \leq k : w_j \models \phi_1 \Rightarrow w_i \models \phi_2$ and $[\delta_j, \delta_i] \models \Delta$

$w \models \phi_1 \rightarrow|_\Delta \phi_2$ if $\exists 0 < j \leq k, w_j \models \phi_1, \forall j < i \leq k, w_i \not\models \phi_2$ and $[\delta_j, \delta_i] \models \Delta$

$w \models \phi_1 U_\Delta \phi_2$ if $\exists 0 < i \leq k, 0 < j \leq k, w_i \models \phi_2$, $[\delta_j, \delta_i] \models \Delta$ and $\forall 0 < j < i, w_j \models \phi_1$

$w \models \phi_1 R_\Delta \phi_2$ if $(\forall 0 < i \leq k, w_i \models \phi_2)$ or $(\exists 0 < i \leq k, w_i \models \phi_1, [\delta, \delta_j] \models \Delta$ and $\forall 0 < j < i, w_j \models \phi_2)$

The intuitive meaning of the TLTL operators is the following. The symbols *true* and *false* stand for the propositional constants true and false, which are satisfied by every timed trace and no trace, respectively. A finite or infinite timed trace satisfies the atomic proposition (a, ψ) if the timed trace is not empty and if there is an action a within ψ. It satisfies $\neg(a, \psi)$ if it does not satisfy (a, ψ). The propositional constructs $\wedge$ and $\vee$ have their usual interpretation as conjunction and disjunction, respectively. The unary operators X_Δ and $\hat{X}_\Delta$ represent next-state operators. Intuitively, the trace w satisfies $X_\Delta\phi$ and $\hat{X}_\Delta\phi$ if w' satisfies ϕ while (δ, δ') satisfies Δ. The only difference between $X_\Delta\phi$ and $\hat{X}_\Delta\phi$ arises when considering the empty trace ε. Indeed, ε satisfies $\hat{X}_\Delta\phi$ but violates $X_\Delta\phi$. Formula $\phi_1 \rightarrow_\Delta \phi_2$ shows a binary necessity, if w satisfies ϕ_1, it must satisfy ϕ_2 during the time (δ, δ') satisfying Δ. $\phi_1 \rightarrow|_\Delta \phi_2$ is the dual formula of $\phi_1 \rightarrow_\Delta \phi_2$, which shows that if w satisfies ϕ_1, it must not

satisfy ϕ_2 during the time (δ,δ') satisfying Δ. Formula $\phi_1 U_\Delta \phi_2$ represents an until property and is satisfied by any timed trace which satisfies ϕ_1 until ϕ_2 becomes valid within Δ. $\phi_1 R_\Delta \phi_2$ is a release formula and is satisfied by any trace which satisfies ϕ_2 unless this formula is released from its obligation by the truth of ϕ_1 during Δ (which might never occur if ϕ_2 =false). Finally, we introduce the derived operators G("generally") and F("eventually") by defining $G_\Delta \phi = falseR_\Delta \phi$ and $F_\Delta \phi = trueU_\Delta \phi$.

5.2 Constructing Timed Processes from TLTL Formulas

We are now ready to show how a TLTL formula can be converted into a timed process, such that the timed traces satisfying the timed process also satisfy the formula.

5.2.1 Constructing Timed Process

Based on previous work on converting temporal logic formulas to automata or transition systems [21,22,23], we now build a timed process $P_\phi = (S, S_0, \rightarrow)$, such that $L_m(P_\phi) \cup L_\omega(P_\phi) \cup L_D(P_\phi)$ is exactly the set of computations satisfying the formula ϕ.

The data structure used as the state names in the set S is the following. Also note that "Clock" is not part of the state name:

– Name: A string that is the name of the state, such as s_1.

– Incoming: The set of incoming edges; each edge is represented by the name of the source node(s) whose outgoing edge leads to the current node.

– New: A set of temporal properties (formulas) that must hold at the current state and have not yet been considered.

– Old: The properties that must hold in the state and have already been considered.

– Next: Temporal properties that must hold as the temporal formulas in the set New of the next state.

– Clock: A set of clocks that decides when the properties should be considered.

The initial states are all in S_0. A state is an initial state if and only if its Incoming set is empty.

$Cl(\phi)$ is the closure of a TLTL formula ϕ which includes all the sub-formula[6] sets of ϕ. $2^{Cl(\phi)}$ is a product of two sets $(A \times L)$ because every

---

[6] Suppose ϕ is a formula of TLTL. A sub-formula of ϕ is defined inductively as follows:

1. ϕ is a sub-formula of ϕ.
2. if $\neg\phi$ is a sub-formula of ϕ for some TLTL formula φ, then so is φ.
3. if $X\phi$ is a sub-formula of ϕ for some TLTL formula φ, then so is φ.
3. if $\alpha \wedge \beta (or\ \alpha \rightarrow_\Delta \beta\ or\ \alpha U_\Delta \beta)$ is a sub-formula of ϕ for some TLTL formula α, β, then so are α and β.

TLTL formula ϕ also includes time constraints.

The clocks of the set C in P_ϕ is are used to measure the time increase between two states. Here, they will also be used to check the timed constraints of the operators. In the timed process P_ϕ, the time constraints of atomic propositions are checked using time sequences, and the time constraints of operators are checked using clocks.

The transition relation $\rightarrow$ is constructed as edges between states according to the following algorithm.

Without losing generality, we may assume that the formula does not contain the operators 'F' and 'G', which can be replaced by formulas using the operator 'U', and that all the negations are pushed inside until they only precede propositional variables.

The algorithm that constructs the transition relation corresponding to ϕ starts with a single node. This node has no incoming edge. Thus, by the end of the construction, a node will be initial iff it contains no incoming edge. It has initially one new obligation in New, namely ϕ, and the sets Old, Next and Clock are initially empty.

With the current node ***N***, the algorithm checks if there are unprocessed obligations left in New. If not, the current node is fully processed and ready to be added to the state set S. If there already is a node in the state set S with the same obligations in both its sets Old and Next, the copy that already exists needs only

to be updated with the new set of incoming edges; the set of incoming edges to the new copy is added to the ones of the old copy.

If no such node exists in state set, then the current node is added to this list, and a new current node is formed for its successor as follows:

- There is initially one edge from ***N*** to the new current node, and this edge is added to the set Incoming of the new current node.
- The set New is set initially to the set Next of ***N***.
- The sets Old, Next and Clock of the new current node are initially empty.

When processing the current node, a formula η in the set New is removed from its list, in the case that η is a proposition. If $\neg\eta$ is in Old, the current node is discarded, as it contains a contradiction. Otherwise, η is added to Old if it is not already there.

When η is not an atomic proposition, the actions on the current node depend on the form of η:

– $\eta = \mathrm{X}_\Delta\varphi$ $(\eta = \hat{\mathrm{X}}_\Delta\varphi)$ Add φ into Next. Set one clock to zero to associate with the Δ of $\mathrm{X}_\Delta\varphi$ $(\hat{\mathrm{X}}_\Delta\varphi)$

– $\eta = \mu \wedge \varphi$ Add both μ and φ to New as the truth of both formula is needed to make η hold.

– $\eta = \mu \vee \varphi$ The node is split into two new nodes, adding μ to New of one new node, and φ to the other. These nodes correspond to the two ways in which η can be made to hold.

– $\eta = \mu U_{\Delta}\varphi$ The node is split into two new nodes: for the first new node, add μ to New and $\mu U_{\Delta}\varphi$ to Next. For the other one, add φ to New and Next and set one clock to zero to associate with the Δ of $U_{\Delta}\varphi$.

– $\eta = \mu R_{\Delta}\varphi$ The node is split into two new nodes: add φ to New of one new node and η to the Next and set one clock to zero to associate with Δ of $R_{\Delta}\varphi$. Then add μ to New of the other new node.

– $\eta = \mu \rightarrow_{\Delta} \varphi$ Add μ to New and φ to Next. Set one clock to zero to associate with Δ of $\rightarrow_{\Delta} \varphi$.

– $\eta = \mu \rightarrow|_{\Delta} \varphi$ Add μ to New and $\neg\varphi$ to Next. Set one clock to zero to associate with Δ of $\rightarrow|_{\Delta} \varphi$.

The idea of splitting nodes is inspired by the CTL model checking graph [24,25,26] when dealing with binary formulas.

The copies are processed in DFS order, i.e., when expansion of the current node and its successors are finished, the expansion of the second copy and its successors is started.

The algorithm is listed as follows in pseudo-code. The function **new_node**() generates a new string for each successive call. The function **negation**(), is defined as follows: **negation**(P_{ϕ}) $= \neg P_{\phi}$, **negation**($\neg P_{\phi}$) $= P_{\phi}$, and similarly for the Boolean constants *true* and *false*. The functions **New***X*(), **Next***X*(),**Incoming***X*(),**Old***X*(),**Clock***X*() ($X = \varepsilon$ represents the current node); $X \in \mathbb{N}$ is an order for the nodes after splitting) are the functions generating the

sets in the splitted nodes. Note that the number of variables in the set ClockX() is not limited, but the clocks are created by two kinds time constraints, ψ and Δ.

```
1    define process_node = [Name: string, Incoming: set of strings, New: set of
2      formulas, Old: set of formulas, Next: set of formulas, Clock: set of clocks];
3    function expansion (Node, Nodes_Set)
4       if New(Node)=∅  then
5         if ∃ND ∈ Nodes_Set with Old(ND)=Old(Node), Next(ND)=Next(Node)
6              Clock(ND)=Clock(Node)
7         then Incoming(ND)=Incoming(ND) ∪ Incoming(Node);
8         return(Nodes_Set)
9         else return(expansion([Name ⇐ New_node(),
10             Incoming ⇐ {Name(Node)}, New ⇐ Next(Node),
11             Olde ⇐ ∅, Next ⇐ ∅, Clock ⇐ ∅],
12             {Node} ∪ Nodes_Set))
13       else
14        let η ∈ New(Node);
15        New(Node):=New(Node)\{η}
16        case η:
17        η = P_φ  or  ¬P_φ  or  η = true  or  η = false
18             if η = false  or negation(η) ∈ Old(Node)
```

19 then return(Nodes_Set)

20 else Old(Node)=Old(Node) $\cup \{\eta\}$

21 return(expansion(Node,Nodes_Set));

22 $\eta = \mathrm{X}_{\Delta}\varphi$ or $\eta = \hat{\mathrm{X}}_{\Delta}\varphi$

23 return(expansion([Name $\Leftarrow$ Name(Node),

24 Incoming $\Leftarrow$ Incoming(Node), New $\Leftarrow$ New(Node),

25 Old $\Leftarrow$ Old(Node) $\cup \{\eta\}$, Next $\Leftarrow$ Next(Node) $\cup \{\varphi\}$,

26 $\Leftarrow$ Clock(Node) $\cup \{c_{\Delta}\}$])

27 $\eta = \mu \wedge \varphi$

28 return(expansion([Name $\Leftarrow$ Name(Node),

29 Incoming $\Leftarrow$ Incoming(Node),

30 New $\Leftarrow$ New(Node) $\cup \{\mu, \varphi\}$\Old(Node), Old $\Leftarrow$ Old(Node) $\cup \{\eta\}$,

31 Next $\Leftarrow$ Next(Node), Clock $\Leftarrow$ Clock(Node)], Nodes_Set))

32 $\eta = \mu \vee \varphi$

33 Node1:=[Name $\Leftarrow$ new_node(), Incoming $\Leftarrow$ Incoming(Node),

34 New $\Leftarrow$ New(Node) $\cup$ ({New1(μ)}\Old(Node),

35 Old $\Leftarrow$ Old(Node) $\cup \{\eta\}$, Next $\Leftarrow$ Next(Node) }

36 Clock $\Leftarrow$ Clock(Node)]

37 Node2:=[Name $\Leftarrow$ new_node(), Incoming $\Leftarrow$ Incoming(Node),

38 New $\Leftarrow$ New(Node) $\cup$ ({New2(φ)}\Old(Node),

39 Old $\Leftarrow$ Old(Node) $\cup \{\eta\}$, Next $\Leftarrow$ Next(Node),

```
40              Clock ⇐ Clock(Node)]
41          return(expansion(Node2,expansion(Node1,Nodes_Set)));
42      η = μU_Δφ
43          Node1:=[Name ⇐ new_node(), Incoming ⇐ Incoming(Node),
44              New ⇐ New(Node) ∪ ({New1(μ)}\Old(Node),
45              Old ⇐ Old(Node) ∪ {η}, Next ⇐ Next(Node) ∪ {Next1(η)}
46              Clock ⇐ Clock(Node)]
47          Node2:=[Name ⇐ new_node(), Incoming ⇐ Incoming(Node),
48              New ⇐ New(Node) ∪ ({New2(φ)}\Old(Node),
49              Old ⇐ Old(Node) ∪ {η}, Next ⇐ Next(Node) ∪ {Next1(φ),
50              Clock ⇐ Clock(Node) ∪ {c_Δ}]
51          return(expansion(Node2,expansion(Node1,Nodes_Set)));
52      η = μR_Δφ
53          Node1:=[Name ⇐ new_node(), Incoming ⇐ Incoming(Node),
54              New ⇐ New(Node) ∪ ({New1(μ)}\Old(Node),
55              Old ⇐ Old(Node) ∪ {η}, Next ⇐ Next(Node) ∪ {Next1(η)}
56              Clock ⇐ Clock(Node)]
57          Node2:=[Name ⇐ new_node(), Incoming ⇐ Incoming(Node),
58              New ⇐ New(Node) ∪ ({New2(φ)}\Old(Node),
59              Old ⇐ Old(Node) ∪ {η}, Next ⇐ Next(Node),
60              Clock ⇐ Clock(Node) ∪ {c_Δ}]
```

```
61            return(expansion(Node2,expansion(Node1,Nodes_Set)));
62       η = μ →_Δ φ
63            return(expansion([Name ⇐ Name(Node),
64            Incoming ⇐ Incoming(Node), New ⇐ New(Node) ∪ { μ },
65            Old ⇐ Old(Node) ∪ { η }, Next ⇐ Next(Node) ∪ { φ }
66            Clock ⇐ Clock(Node) ∪ { c_Δ }]
67       η = μ →|_Δ φ
68            return(expansion([Name ⇐ Name(Node),
69            Incoming ⇐ Incoming(Node), New ⇐ New(Node) ∪ { μ },
70            Old ⇐ Old(Node) ∪ { η }, Next ⇐ Next(Node) ∪ { ¬φ }
71            Clock ⇐ Clock(Node) ∪ { c_Δ }])
72   end expansion;
73   main function create_node ( φ )
74       return(expansion([Name ⇐ new_node(),Incoming ⇐ {init},
75            New ⇐ { φ }, Old ⇐ ∅ , Next ⇐ ∅ , Clock ⇐ ∅ ], ∅ ))
76   End create_node;
```

Figure 2 offers a partial example of the algorithm. It is immediate from the algorithm that builds the transition relation that the timed process allows finite

timed traces compatible with the initial TLTL formula. Thus, we have the following theorem. The timed process P_{ϕ}^{F} is the one constructed according to the algorithm presented above.

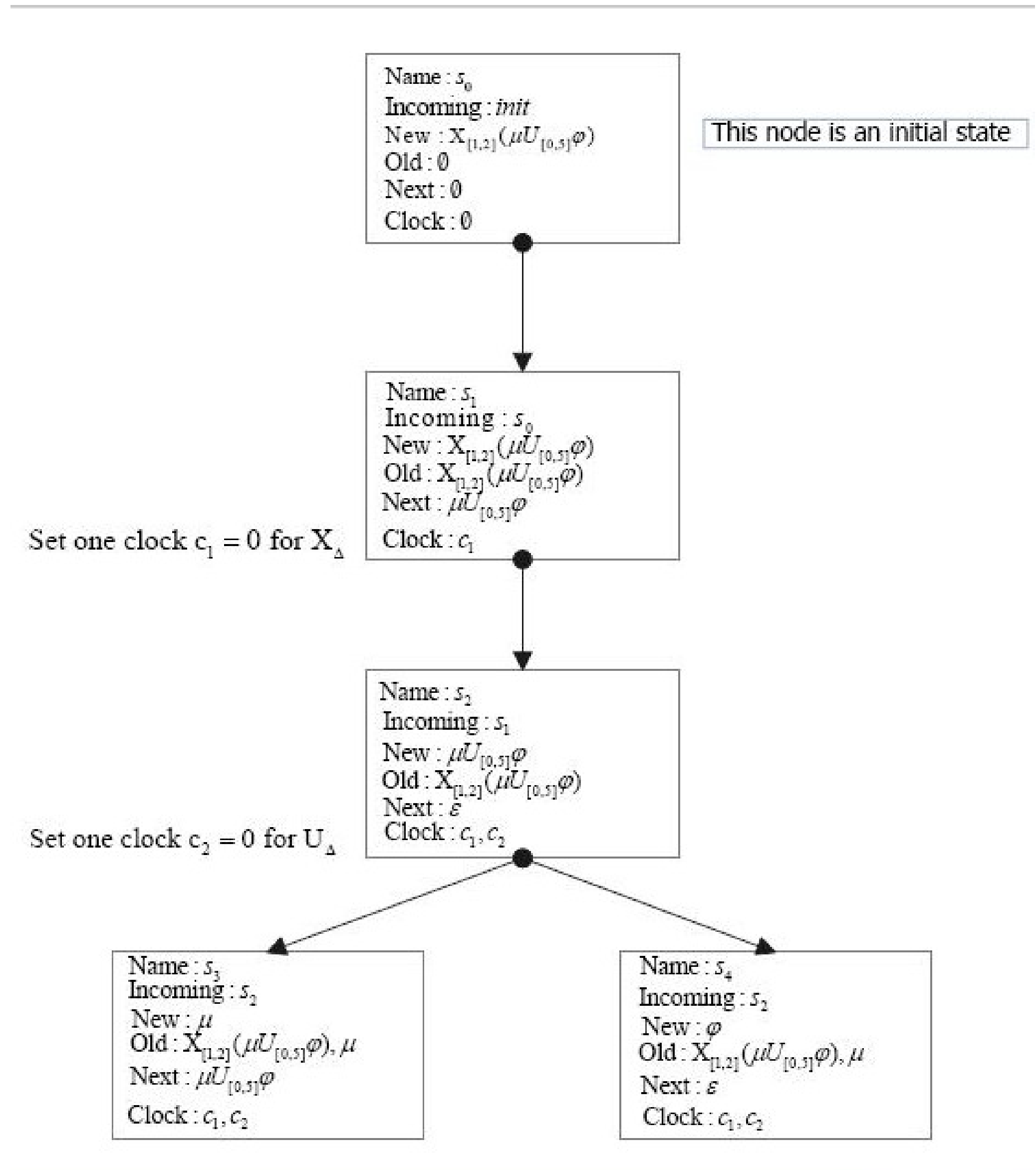

Fig. 2. An example for the algorithm

Theorem 3 *Let ϕ be a TLTL formula. Then there exists a timed process P_ϕ^F such that $w \models \phi$ if and only if $w \in L_f(P_\phi^F)$, for any $w \in (A \times L)^*$*

In the remainder of this chapter, we show how to generate a timed process P_ϕ satisfying $w \models \phi$ if and only if $w \in L_m(P_\phi) \cup L_\omega(P_\phi) \cup L_D(P_\phi)$, for any $w \in (\mathrm{A} \times \mathrm{L})^* \cup (\mathrm{A} \times \mathrm{L})^\omega$ We show this by separating the domains to which w belongs.

5.2.2 Allowing Finite Maximal Timed Traces

In the first step we show how to generate a timed process P_ϕ^M satisfying $w \models \phi$ if and only if $w \in L_m(P_\phi)$, for any $w \in (\mathrm{A} \times \mathrm{L})^*$. The basic approach relies on altering the timed process P_ϕ^F to handle maximal timed traces. More precisely, for every state s in P_ϕ^F, we check if all formulas contained in the New set are satisfied by the trace *ε*. Checking for acceptance of the trace *ε* can be done syntactically in the algorithm along the structure of the formulas. Then, for every state s in P_ϕ^F such that each TLTL formula ϕ labeling the New set in s is satisfied by *ε,* we add a transition $s \xrightarrow{\tau} s_\varepsilon$, where s_ε is a new state with the New set labeled with $\{\hat{\mathrm{X}}\mathrm{false}\}$, and for which the set New and Next only contains *ε*. We use such a state to distinguish from other states also having no outgoing transitions. Note that s_ε is a deadlock state, but it differs from the some other

deadlock states to allow the identification of maximal time traces.[7]

Theorem 4 *Let ϕ be a TLTL formula. Then there exists a timed process P_ϕ^M such that $w \models \phi$ if and only if $w \in L_m(P_\phi^M)$, for any $w \in (A \times L)^*$*

Proof

Let P_ϕ^M be constructed using the algorithm above.

$\Rightarrow$: According to the algorithm outlined above, if $w \models \phi$, $w \in (\mathrm{A} \times \mathrm{L})^*$ and ϕ is the formula P_ϕ^M is constructed from, then $s_0 \overset{w}{\Rightarrow} s \overset{\tau}{\rightarrow} s_\varepsilon$ holds. That is $s_0 \overset{w}{\Rightarrow} s_\varepsilon \not\rightarrow$. Thus, w is a maximal trace in P_ϕ^M. Recall Definition 4 in Chapter 3, and we have $w \in L_m(P_\phi^M)$.

$\Leftarrow$: If $w \in L_m(P_\phi^M)$ for any $w \in (\mathrm{A} \times \mathrm{L})^*$, there exits $s_0 \overset{w}{\Rightarrow} s \not\rightarrow$ in P_ϕ^M (Definition 4). According to the algorithm of P_ϕ^M, there exits $s_0 \overset{w}{\Rightarrow} s \overset{\tau}{\rightarrow} s_\varepsilon$. Thus, $w \models \phi$. (ϕ is the formula P_ϕ^M is constructed from) □

Theorem 4 is a consequence of the fact that our construction ensures that $w \in L_m(P_\phi^M)$ if and only if $s \overset{w}{\Rightarrow}_{P_\phi^M} s_\varepsilon$ and that $s \overset{w}{\Rightarrow}_{P_\phi^M} s_\varepsilon$ holds if and only if $w \models \phi$.

[7] Other deadlock states such as the divergent state, the set New and Next are empty and Incoming set contains the current node.

5.2.3 Allowing ω-final Timed Traces

The states in the timed process P_{ϕ}^{F} we constructed are labeled by sets of formulas. The construction of this timed process, or more precisely, the algorithm of the transition relation already ensures that infinite timed traces emanating from a state are guaranteed to satisfy each formula labeling this state. Recall the definition of ω-final states from Chapter 3 (Definition 5). To alter the timed process P_{ϕ}^{F} to handle ω-final timed traces, for every state s in P_{ϕ}^{ω} we check whether all formulas contained in the set New of s are satisfied by the ω-final condition. If so, we reduce all the same recurrences to one ω-final state. Note that the ω-final condition is similar but still differs from the traditional Muller acceptance [12] as long as we try to build timed process from TLTL formula. For example, in a time-action sequence, the action sequence may look like this: abacbadbcababdcdcadb… [8] . This sequence satisfies $((a \rightarrow_{[1,2]} b) \wedge (b \rightarrow_{[1,2]} a)) \wedge ((c \rightarrow_{[1,2]} d) \wedge (d \rightarrow_{[1,2]} c))$ provided that the time sequences of the actions satisfy the time constraints. The New set in ω-final state then contains $\{(a \rightarrow_{[1,2]} b), (b \rightarrow_{[1,2]} a), (c \rightarrow_{[1,2]} d), (d \rightarrow_{[1,2]} c)\}$.

We have the following results.

Theorem 5 *Let* ϕ *be a TLTL formula. Then there exists a timed process* P_{ϕ}^{ω}

[8] The whole action sequence is not a repeating run as accepted by Muller acceptance [9], We prefer to call ab and cd recurrences rather than runs. As a matter fact b occurs after a, and then another a occurs after b…; c,d do the same, but there is no order between ‘a,b’ and ‘c,d’.

such that $w \models \phi$ *if and only if* $w \in L_\omega(P_\phi^\omega)$ *, for any* $w \in (A \times L)^\omega$

Theorem 6 *Let p be a convergent timed process, and let* ϕ *be a TLTL formula. Then* $p \models \phi$ *if and only if* $L_m(p) \subseteq L_m(P_\phi^M)$ *and* $L_\omega(p) \subseteq L_\omega(P_\phi^\omega)$

5.2.4 Allowing Divergent Timed Traces

As a third step, we build a timed process P_ϕ^D that additionally takes divergent timed traces of timed processes into account. We modify P_ϕ to a timed process P_ϕ^D by adding divergent states. According to the definition of divergence, the divergent states of P_ϕ^D should have the following property; If $w \in (\mathrm{A} \times \mathrm{L})^*$ such that $w \cdot w' \models \phi$ for any $w' \in (\mathrm{A} \times \mathrm{L})^* \cup (\mathrm{A} \times \mathrm{L})^\omega$, then the states reachable in P_ϕ^D via w should be divergent. Essentially, divergence is intended to capture tautologies. The construction of P_ϕ^D relies on the construction of a timed process for recognizing words in $(\mathrm{A} \times \mathrm{L})^*$, which is done by P_ϕ. Thus, for each state s in P_ϕ, we check whether the formula $\vee_{F \in l(s)} \wedge_{\phi \in F} \phi$ is a tautology, where $l(s)$ is the set of sets of formulas labeling the set New of s in the timed process. We check every formula labeling the set New of s until we find a tautology formula; this makes s a divergent state and also makes the formula $\vee_{F \in l(s)} \wedge_{\phi \in F} \phi$ a tautology. If so, we make this state s divergent by adding a τ-loop edge to it and eliminating any other edge to any other state. Note that the tautology check can be performed algorithmically [27]

Lemma 2 *Let s be the start state of the timed process* P_ϕ^D *, and let* $w \in (\mathrm{A}\times\mathrm{L})^*$ *such that* $s \Uparrow_{P_\phi^D} w$. *Then* $w \cdot w' \models \phi$*, for any* $w' \in (\mathrm{A}\times\mathrm{L})^* \cup (\mathrm{A}\times\mathrm{L})^\omega$.

Proof **:**

Since $s \Uparrow_{P_\phi^D} w$, $w \in (\mathrm{A}\times\mathrm{L})^*$, and $w' \in (\mathrm{A}\times\mathrm{L})^* \cup (\mathrm{A}\times\mathrm{L})^\omega$, it holds that $s \xrightarrow{w} s_D \xrightarrow{w'}$, where s_D is a divergent state. According to the algorithm, we already capture the tautologies at s_D, since *w'* is a divergent trace. Thus, $w \cdot w' \models \phi$. (Note that, we don't care how *w'* looks like, but since it is a timed trace, $w' \in (\mathrm{A}\times\mathrm{L})^* \cup (\mathrm{A}\times\mathrm{L})^\omega$ always holds.) □

Since timed process P_ϕ^M and P_ϕ^ω are based on P_ϕ , and the algorithms do not conflict with the one in P_ϕ^D , We can extend Lemma 2 to an infinite prefix.

Lemma 3 *Let s be the start state of timed process* P_ϕ^D *, and let* $w \in (\mathrm{A}\times\mathrm{L})^* \cup (\mathrm{A}\times\mathrm{L})^\omega$ *such that* $s \Uparrow_{P_\phi^D} w$. *Then* $w \cdot w' \models \phi$*, for any* $w' \in (\mathrm{A}\times\mathrm{L})^* \cup (\mathrm{A}\times\mathrm{L})^\omega$

Theorem 7 Let *p* be a timed process, and let ϕ be a TLTL formula such that $p \models \phi$. Then $w \in L_D(p)$ implies $w \in L_D(P_\phi^D)$.

The proof of this theorem follows from the fact that if $w \in L_D(p)$, then there

exits a finite prefix *w'* of *w* such that $w'\cdot w'' \in L_D(p)$.This implies that *w'* must lead to a divergent state in P_ϕ^D.

Proof of Theorem 7

$p \models \phi$ implies that $w \models \phi$ for every trace *w* of p. For every divergent timed trace $w_D \in L_D(p)$, it also holds that $w_D \models \phi$. Then, we can construct a process P_ϕ^D from ϕ to allow all the divergent timed trace w_D such that $w_D \models \phi$. The rest of the proof is then similar with the proof of Theorem 4. Thus, $w_D \in L_D(P_\phi^D)$. □

5.2.5 Allowing all Timed Traces

Now with the timed processes P_ϕ^M, P_ϕ^ω, P_ϕ^D, it is easy to construct a P_ϕ^T based on P_ϕ combining all the algorithms in P_ϕ^M, P_ϕ^ω, P_ϕ^D. This construction leads to the following result.

Theorem 8 Let $w \in (\mathrm{A}\times\mathrm{L})^* \cup (\mathrm{A}\times\mathrm{L})^\omega$. Then $w \models \phi$ if and only if $w \in L_m(P_\phi^T) \cup L_\omega(P_\phi^T) \cup L_D(P_\phi^T)$

The validity of this theorem is taken care of by Theorem 4, Theorem 5 and Lemma 3 when considering that P_ϕ^T possesses by construction the same maximal timed traces and the same ω-final timed traces and the same divergent timed traces.

Theorem 7 and Theorem 8 are the keys to proving the following theorem, which extents Theorem 6 to arbitrary timed processes.

Theorem 9 Let q be a timed process and ϕ be a TLTL formula. Then there exists a timed process P_ϕ^T such that $q \models \phi$

if and only if (Ⅰ) $L_D(q) \subseteq L_D(P_\phi^T)$

(Ⅱ) $L_F(q) \setminus L_D(q) \subseteq L_F(P_\phi^T)$

(Ⅲ) $L_m(q) \setminus L_D(q) \subseteq L_m(P_\phi^T)$

(Ⅳ) $L_\omega(q) \setminus L_D(q) \subseteq L_\omega(P_\phi^T)$

Proof

For the $\Rightarrow$ direction, let $q \models \phi$, i.e. $w \models \phi$ for all $w \in L_M(q) \cup L_\omega(q) \cup L_D(q)$. By Theorem 8 we also have $w \in L_M(P_\phi^T) \cup L_\omega(P_\phi^T) \cup L_D(P_\phi^T)$. We distinguish the following cases.

Case 1: $w \in L_D(q)$ This case is established by Theorem 7.

Case 2: $w \in L_F(q) \setminus L_D(q)$ Since q is a timed process, $w \in L_F(q)$ is always a finite prefix of a maximal trace or an infinite trace. Thus, we may conclude the existence of some $w' \in (A \times L)^* \cup (A \times L)^\omega$ such that $w \cdot w' \in L_M(P_\phi^T) \cup L_\omega(P_\phi^T)$ $\cup L_D(P_\phi^T)$. By construction, every trace which $w \cdot w'$ that reaches a divergent or $\omega-$final state s in P_ϕ^T still satisfies $L_M(s') \subseteq (\mathrm{A} \times \mathrm{L})^*$ (s' is the state before s),

w is therefore a finite prefix for $L_M(P_\phi^T) \cup L_\omega(P_\phi^T) \cup L_D(P_\phi^T)$. Thus, $w \in L_F(P_\phi^T)$, and $L_F(q) \setminus L_D(q) \subseteq L_F(P_\phi^T)$, as desired.

Case 3: $w \in L_M(q) \setminus L_D(q)$, $w \in (\mathrm{A} \times \mathrm{L})^*$ and by Theorem 8 we have $w \in L_M(P_\phi^T)$, and $L_M(q) \setminus L_D(q) \subseteq L_M(P_\phi^T)$, as desired.

Case 4: $w \in L_\omega(q) \setminus L_D(q)$, $w \in (\mathrm{A} \times \mathrm{L})^\omega$ and by Theorem 8, we have $w \in L_\omega(P_\phi^T)$, and $L_\omega(q) \setminus L_D(q) \subseteq L_\omega(P_\phi^T)$, as desired.

For the $\Leftarrow$ direction, assume that $q \not\models \phi$, i.e., $\exists w \in L_M(q) \cup L_\omega(q) \cup L_D(q)$: $w \not\models \phi$. By Theorem 8 we also know that $w \notin L_M(P_\phi^T), w \notin L_\omega(P_\phi^T)$, and $w \notin L_D(P_\phi^T)$. We distinguish the following cases.

Case 1: $w \in L_D(q)$ Then $w \in L_D(P_\phi^T)$ which contradicts $w \notin L_D(P_\phi^T)$.

Case 2: $w \in L_M(q) \setminus L_D(q)$ Then $w \in L_M(P_\phi^T)$ which contradicts $w \notin L_M$ (P_ϕ^T).

Case 3: $w \in L_\omega(q) \setminus L_D(q)$ Then $w \in L_\omega(P_\phi^T)$ which contradicts $w \notin L_\omega(P_\phi^T)$.

Thus, direction "$\Leftarrow$" holds, as desired. □

5.3 Relating TLTL Satisfaction to the Timed Must-preorder

As the last step in relating the TLTL satisfaction relation $\models$ to the timed must-preorder $\sqsubseteq_T^{Must}$,we transform P_ϕ^T into a purely nondeterministic timed process T_ϕ while preserving all languages as outlined in Chapter 4. Thus, Theorem 2 is valid for T_ϕ as well as P_ϕ^T. By combining Theorems 2 and 8 we

obtain the desired main result:

Corollary 1: Timed Must-Testing and TLTL Model Checking. *Let q be a timed process and ϕ be a TLTL formula. Then there exists a timed process T_ϕ such that $q \models \phi$ if and only if $T_\phi \sqsubseteq_T^{must} q$.*

As a consequence of this corollary, our notion of timed must-testing not only extends DeNicola and Hennessy's must-preorder to timed processes, but is also compatible with the satisfaction relation of linear-time temporal logic with time constraints.

6 Parallel composition

A system specification or a protocol can be defined algebraically or logically, and usually consists of different parts of the specification mixing algebraical and logical design. It is important to consider a method to assemble all the parts into a general specification [5]. Indeed, all our previous effort remains of limited use without such a method. In this chapter, we introduce an operator on timed processes for merging the different parts of the real-time system or protocol: parallel composition.

Our parallel composition operator $\underset{A'\times L'}{\|}$, where $A'\times L' \subseteq A\times L$, is inspired by the interface parallel operator of CSP [28]. However, our actions have time components, that is, the events appearing in the common interface $A'\times L'$ probably have different time delays in the processes being composed. To solve this problem, we simply regard the longer delays as the delays of the common actions while we merge two time processes. In other words, the common action occurring in one process should wait for the same one in the other process. Now the question that naturally arises concerns the interpretation of timed traces. We adopt the following point of view: Intuitively, the "fair merge" of the finite traces (or ω-final traces) of timed processes p and q should form the finite trace (or ω-final trace) of the timed process $p \underset{A'\times L'}{\|} q$. Moreover, an ω-final trace of one

timed process, when merged with a finite trace of the other timed process, should result in an ω-final trace of the timed process $p \underset{A'\times L'}{||} q$.

Note that the failure of a merge (i.e., the deadlock of the parallel composition operator), does not include only the usual deadlock of the interface parallel, but also the deadlock caused by the (lengthened) delays failing to satisfy their own timed constraints (which might have been satisfied before merging).

Formally, we define the parallel composition of two timed processes $((A_p \times L_p) \cup \{\tau\}, C_p, P, \rightarrow_p, p_0)$ and $((A_q \times L_q) \cup \{\tau\}, C_q, Q, \rightarrow_q, q_0)$ to be the timed process $((A_{p \underset{A'\times L'}{||} q} \times L_{p \underset{A'\times L'}{||} q}) \cup \{\tau\}, C_p \cup C_q, P \underset{A'\times L'}{||} Q, \rightarrow_{p \underset{A'\times L'}{||} q}, p_0 \underset{A'\times L'}{||} q_0)$, where $P \underset{A'\times L'}{||} Q = \{p' \underset{A'\times L'}{||} q' \mid p' \in P, q' \in Q\} \cup \{q' \underset{A'\times L'}{||} p' \mid p' \in P, q' \in Q\}$. The transition relation $\rightarrow_{p \underset{A'\times L'}{||} q}$ is the least relation satisfying the following rules.

(1) $p' \underset{\Phi c_p}{\overset{(a,\delta_p)}{\rightarrow_p}} p''$ and $q' \underset{\Phi c_q}{\overset{(a,\delta_q)}{\rightarrow_q}} q''$ implies $p' \underset{A'\times L'}{||} q' \xrightarrow[\Phi c_p \cap \Phi c_q]{(a, Max(\delta_p, \delta_q))}_{p||q} p'' \underset{A'\times L'}{||} q''$ if $L(p') \in (A \times L)^\omega$

(2) $p' \underset{\Phi c_p}{\overset{(a,\delta_p)}{\rightarrow_p}} p''$ and $q' \underset{\Phi c_q}{\overset{(a,\delta_q)}{\rightarrow_q}} q''$ implies $p' \underset{A'\times L'}{||} q' \xrightarrow[\Phi c_p \cap \Phi c_q]{(a, Max(\delta_p, \delta_q))}_{p||q} q'' \underset{A'\times L'}{||} p''$ if $L(q') \notin (A \times L)^\omega$

These rules are in accordance with our above-mentioned intuition of system behavior. Rule (2) , the "switching" of the states of p and q allows us to fairly merge ω-final traces with ω-final traces and also ω-final traces with finite timed traces. One can now obtain the timed may- and must- preorders with respect to the new operators. Note in passing that our fair merge is based on the operation of

concatenation of well-behaved ω-language [13]. Indeed, all our timed trace languages are well-behaved, as we exclude Zeno behavior.

Theorem 10 Let p_1, p_2, q_1 and q_2 be timed processes and $A' \times L' \subseteq A \times L$. Then

(i) $p_1 \sqsubseteq_T^{may} p_2$ and $q_1 \sqsubseteq_T^{may} q_2$ implies $p_1 \underset{A' \times L'}{\|} q_1 \sqsubseteq_T^{may} p_2 \underset{A' \times L'}{\|} q_2$

(ii) $p_1 \sqsubseteq_T^{must} p_2$ and $q_1 \sqsubseteq_T^{must} q_2$ implies $p_1 \underset{A' \times L'}{\|} q_1 \sqsubseteq_T^{must} p_2 \underset{A' \times L'}{\|} q_2$

The proof of this theorem is an immediate consequence of the characterizations of timed may- and must- preorders and our extension results, as presented in Chapter 4.

Proof of Theorem 10: (i) According to Definitions 7, 9, and 10, set t'and t" for p_1, p_2 and q_1, q_2, respectively. By $p_1 \sqsubseteq_T^{may} p_2$ and $q_1 \sqsubseteq_T^{may} q_2$, $p_1 \text{ may}_T\ t'$ and $p_2 \text{ may } t''$ have the same computation *c'*, also $q_1 \text{ may}_T\ t''$ and $q_2 \text{ may } t''$ have the same computation *c"*. By the definition of parallel composition operator, they all execute the same common action(s) with same delays (the longer one) included in *c'* and *c"*. According to the definition of partial computation, the computation of the common action(s) is same. Thus, $p_1 \underset{A' \times L'}{\|} q_1$ and $p_2 \underset{A' \times L'}{\|} q_2$ have the same computation under same test. Therefore, $p_1 \underset{A' \times L'}{\|} q_1 \sqsubseteq_T^{may} p_2 \underset{A' \times L'}{\|} q_2$ as desired. Item (ii) is proved similarly with must_T. □

Regarding finite timed traces, one can then adapt the corresponding proofs of DeNicola and Hennessy [10]. The parallel composition operator is a consequence of the formalization of our intuition of fair merging.

7 Motivating Example

As motivation for the work in this thesis, consider the design of a very simple boarding system as shown in Fig.7.1.

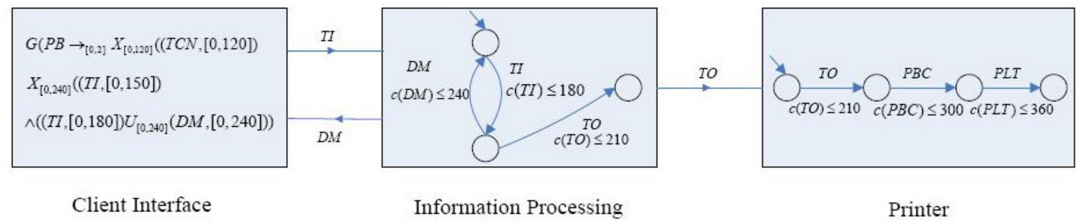

Fig. 7.1 A simple boarding system

The architecture of the system has already been fixed by some system designers and consists of an input interface (Client Interface), a medium (Information Processing), and an output interface (Printer). These are shown in the figure, where the abbreviations represent the following actions:

PB = push button

TCN = type client number

TI = transfer input data

DM = display message

TO = transfer output data

PBC = print boarding card

PLT = print luggage tag

The components communicate with the system's environment and among themselves via actions. The actions connecting components are regarded as common actions, such as TI and DM. Each component in turn has its own specification. In this particular case, the Information Processing and the Printer are given as timed process, reflecting the fact that their design is relatively advanced. The Client Interface is however specified assertionally by a TLTL formula, that is, on a more abstract level.

The formula states that whenever a client pushes the button and types the client number in 120 time units during the sequence of the Client Interface, the remainder of the execution must begin with a sequence of transfer input data actions within 30 time units followed by a display message action in no less than 240 time units, or with a transfer output data action within 30 time units.

Finally, the overall specification of the system's required behavior may be given by the following TLTL formula.

$$Spec = G(PB \rightarrow_{[0,2]} X_{[0,120]}(TCN,[0,120]) \rightarrow (F_{[0,120]}((PBC,[0,300]) \rightarrow X_{[0,60]}(PBC ,[0,360]))))$$

This formula encodes a certain reliability guarantee of the system regarding the eventual delivery of the client information. More precisely, it dictates that in any sequence of actions which the system performs, whenever a client pushes the

button and types in the numbers, two print actions eventually follow within their proper time constraints. Now the obvious question that a designer would be interested in is whether the specification of the Client Interface is "strong enough" to ensure that the system satisfies Spec. To demonstrate that the TLTL specification of the Client Interface is strong enough to ensure that the system is correct, in the sense of satisfying the TLTL formula Spec given above, we may use the results of this paper as follows.

- Construct the purely nondeterministic timed process T_{Spec} for TLTL formula Spec, as illustrated in Chapter 5.
- Construct the purely nondeterministic timed process T_{client} for TLTL formula ϕ_{client} describing the behavior of the client interface as showed in Fig 7.1.
- Assemble the overall system:

  $System = T_{client} \underset{\{TI,DM\}}{||}$ Information Processing $\underset{\{TO\}}{||}$ Printer
- Determine whether $T_{Spec} \sqsubseteq_{T}^{Must} System$

In this case, the answer is positive. Theorem 10 guarantees that replacing T_{client} with any timed process p such that $T_{cient} \sqsubseteq_{T}^{must} p$ will ensure that the overall system meets its specification. If p is a timed process then $T_{cient} \sqsubseteq_{T}^{must} p$ holds exactly when $p \models \phi_{client}$. One example of such a p is shown in Fig 7.2.

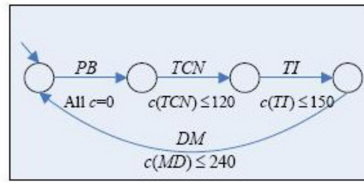

Fig 7.2 Refinement of Client Interface

The motivating example we proposed above is a specification of a boarding system. Since it is a system specification, we do not have to deal with time delays in this particular case. As seen in the example, we have the choice of specifying parts of the system logically (such as the Client Interface), and other parts algebraically (such as Information Processing and Printer). As we mentioned at the beginning of the chapter, the architecture of the system has already been fixed by some system designers and some parts are readily defined algebraically using timed transition systems such as timed processes, but others are easier to express logically using a logical language like TLTL. Our theory is not only applied on unifying different parts of one specification, but is also able to make the implementation coincide with the specification or the test case generated from it while the two might be defined logically and algebraically, respectively.

8 Conclusions

We proposed in this paper a model of timed processes based on timed transition systems [29]. 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 Muller automata [12]. We also extended the testing theory of De Nicola and Hennessy [10] to timed testing. We illustrated that timed processes provide a uniform basis for analyzing heterogeneous reactive-system specifications given as a mixture of timed labelled transition systems [29] and formulas in linear-time temporal logics [11,15,16,20] with timed constraints (TLTL—LTL with timed constraints). We then studied the derived timed may and must preorders and developed alternative characterizations. 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.

We also showed that the timed must-preorder degrades to a variant of reverse timed trace inclusion when its first argument is purely nondeterministic. Using this result, we established the standard algorithms for constructing timed processes from TLTL formulas which can be adapted to our setting in such a way

that TLTL model checking reduces to checking our form of timed trace inclusion. We provided a uniform basis for analyzing heterogeneous real-time system specifications with a mixture of timed transition systems and linear-time temporal logic (LTL) formulas with time constraints. To illustrate the utility of our novel framework, we presented several operators for constructing specifications, argued that the timed must-preorder is substitutive for the operators, and gave an example showing how they may be used to help building system design.

The significance of our results stems from two facts. The first one is that while algorithms and techniques for real-time testing have been studied actively [30,31], 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). This paper offers the first step on bringing logic and algebraic timed specifications together, thus obtaining heterogeneous specifications for real-time systems, that combine the advantages of the two paradigms.

9 Open problems

This thesis is just a start for developing techniques combining operational and assertional styles of timed specifications. The studying of techniques mixing operators from timed process algebras and TLTL is a widely open area.

This thesis establishes standard algorithms for constructing timed processes or timed transition systems from TLTL formulas. How to go the other way around is still open for research.

The timed preorder testing developed from DeNicola and Hennessy's preorder testing is not the only testing relation. Other testing relations with the addition of time constraints will also be exciting to investigate.

We set up a testing framework and characterizations which can be regarded as general test cases. However, we did not care about the test generation. For sure, this will also be an interesting issue for discussion.

References

1 Cohn, A.: The notion of proof in hardware verification. J. Autom. Reasoning 5 (1989) 127–139

2 Bertolino, A.: Software testing research: Achievements, challenges, dreams. In: Future of Software Engineering, IEEE (2007) 85–103

3 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)

4 Tretmans, J.: A formal approach to conformance testing. In: Protocol Test System, VI, Elsevier (1994) 257–276

5 Lamport, L.: Specifying Systems. Addison-Wesley (2002)

6 Tretmans, J.: Conformance testing with labelled transition systems: implementation relations and test generation. Computer Networks and ISDN Systems 29 (1996) 49–79

7 De Nicola, R., Hennessy, M.: Testing equivalences for processes. Theoretical Computer Science 34 (1983) 83–133

8 Hoare, C.: Communicating sequential processes. Prentice Hall (1985)

9 Harry R.Lewis and Christos H. Papadimitriou, Elements of the theory of computation, Prentice-Hall,(1998) 2nd ed.

10 De Nicola, R., Hennessy, M.: Testing equivalences for processes. Theoretical Computer Science 34 (1983) 83–133

11 Alur, R., Courcoubetis, C., Dill, D.: Model-checking for real-time systems. In: Proceedings of the 5th Annual Symposium on Logic in Computer Science, IEEE Computer Society Press (1990) 414–425

12 Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126

(1994) pp.183–235

13 Bruda, S.D., Akl, S.G.: Real-time computation: A formal definition and its applications. International Journal of Computers and Applications 25:4 (2003) pp.247–257

14 Abramsky, S.: Observation equivalence as a testing equivalence. Theoretical Computer Science 53 (1987) pp.225–241

15 E. M. Clarke, Orna Grumberg, Doron Peled, Model Checking., MIT Press (2000)

16 C. Baier, JP Katoen, K. Larsen, Principles of Model Checking, MIT Press, (2008)

17 Timo Latvala, Automata-theoretic and bounded model checking for linear temporal logic, Research Report A95, Helsinki University of Technology, Laboratory for Theoretical Computer Science, Espoo, Finland (2005)

18 Leslie Lamport, Real-Time Model Checking is Really Simple, Springer Berlin / Heidelberg (2005)

19 M.Y.Vardi, P.Wolper, Reasoning about Infinite Computations, Academic Press, Inc. (1994)

20 P.Bellini, R.Mattolini, and P. Nesi, Temporal logics for real-time system specification, ACM Computing Surveys 32:1(2000), pp.12-42

21 R.Gerth, D.Peled, M.Y.Vardi, P.Wolper, Simple On-the-fly Automatic Verification of Linear Temporal Logic, in Proceedings of the IFIP symposium on Protocol Sepcification, Testing and Verification (PSTV' 95), Warsaw, Poland, (1995) pp.3-18

22 M.Vardi and P.Wolper, An automata-theoretic approach to automatic program verification, in First Annual IRRR Symposium on Logic in Computer Science (LICS '86), (1986) pp.332-344

23 A. Fantechi, S. Gnesi, G. Ristori, Modelling Transition Systems within an Action Based Logic, ERCIM technical report 1995-B4-49-12 (1995)

24 Alur, R.; Courcoubetis, C.; Dill, D. Model-Checking for Real-time Systems., Fifth IEEE Symposium on Logic in Computer Science. (LICS), pp. 414–425, (1990)

25 A. Bouajjani, S. Tripakis, S. Yovine, On-the-fly symbolic Model Checking for Real-Time Systems, IEEE RTSS'97, (1997) pp.25

26 R.F.Lutje Spelberg and W.J.Toetenel, Real-Time Model Checking based on Splitting, Nordic Journal of Computing 8 (2001), pages 88 - 120.

27 Cong.J,Perk.J On acceleration of the check tautology logic synthesis algorithmusing an FPGA-based reconfigurable coprocessor , 5th IEEE Symposium on FPGA-based Custom Computing Machines (1997) pp. 246 - 247

28 Steve Schneider, Concurrent and Real Time System, John Wiley & Sons Inc. (1999)

29 T.Henzinger, Z.Manna, A.Pnueli, Temporal Proof Methodologies for Real-Time Systems, CS-TR-91-1383, Stanford University, (1990)

30 Springintveld, J., Vaandrager, F., D'Argenio, P.R.: Testing timed automata. Theoretical Computer Science 254 (2001) 225-257

31 Briones, L.B., Brinksma, E.: A test generation framework for quiescent real-time systems. In: Formal Approaches to Testing of Software. (2004) 71-85