The Power of Reachability Testing for Timed Automata

Luca Aceto[1] *, Patricia Bouyer[2], Augusto Burgueño[3] ** and Kim G. Larsen[1]

[1] **BRICS**[†], Department of Computer Science, Aalborg University, Fredrik Bajers Vej 7-E, DK-9220 Aalborg Ø, Denmark. Email: `{luca,kgl}@cs.auc.dk`, Fax: +45 98 15 98 89

[2] Laboratoire Spécification et Vérification, CNRS URA 2236, Ecole Normale Supérieure de Cachan, 61 av. du Président Wilson, 94235 Cachan Cedex, France. Email: `bouyer@lsv.ens-cachan.fr`, Fax: +33 1 47 40 24 64

[3] ONERA-CERT, Département d'Informatique, 2 av. E. Belin, BP4025, 31055 Toulouse Cedex 4, France. Email: `a.burgueno@acm.org`, Fax: +33 5 62 25 25 93

Abstract. In this paper we provide a complete characterization of the class of properties of (networks of) timed automata for which model checking can be reduced to reachability checking in the context of testing automata.

1 Introduction

The main motivation for the work presented in this paper stems from our practical experience with Uppaal [8], a tool for the verification of behavioural properties of real-time systems specified as networks of timed automata [3]. One of the main design criteria behind Uppaal has been that of efficiency, and its computational engine has originally been restricted to a collection of efficient algorithms for the analysis of simple *reachability* properties of systems. However, in practice one often wants to examine a model to discover whether it enjoys a number of properties that cannot be directly expressed via reachability. Model checking of properties other than plain reachability ones may currently be carried out in Uppaal as follows. Given a property ϕ to model check, the user must provide a *test automaton* T_ϕ for it. The test automaton must be such that the original system S has the property expressed by ϕ precisely when none of the distinguished reject states of T_ϕ can be reached by $S \| T_\phi$, i.e., the agent obtained by making the test automaton interact with the system under investigation. This raises the question of which properties may be analyzed by Uppaal in this manner. In this paper we answer this question by providing a complete characterization of the class of properties of (networks of) timed automata for which model checking can be reduced to reachability testing in the sense outlined above.

* Partially supported by the Human Capital and Mobility project Express, a grant from the Italian CNR, Gruppo Nazionale per l'Informatica Matematica (GNIM), and a visiting professorship at the Laboratoire Spécification et Vérification, Ecole Normale Supérieure de Cachan.

** Partially supported by a Research Grant of the Spanish Ministry of Education and Culture and by BRICS. This work was partially carried out while the author was visiting Aalborg University.

† **B**asic **R**esearch **i**n **C**omputer **S**cience, Centre of the Danish National Research Foundation.

In our previous study [2], we have considered SBLL, a property language suitable for expressing safety and bounded liveness properties of real-time systems. In particular, we have shown that SBLL is *testable*, in the sense that suitable test automata may be derived for any property of SBLL. However, as we shall demonstrate in this paper, SBLL is not *expressive complete* with respect to reachability testing because it cannot express all the properties for which test automata can be derived. As the main result of this paper, we present an extension, $L^{-}_{\forall S}$, of SBLL which is shown to characterize exactly the limit of the testing approach. More precisely we show that:

– every property ψ of $L^{-}_{\forall S}$ is testable, in the sense that there exists a test automaton T_ψ such that S satisfies ψ if and only if $S \| T_\psi$ cannot reach a reject state, for every system S; and
– every test automaton T is expressible in $L^{-}_{\forall S}$, in the sense that there exists a formula ψ_T of $L^{-}_{\forall S}$ such that, for every system S, the agent $S \| T$ cannot reach a reject state if and only if S satisfies ψ_T.

This expressive completeness result will be obtained as a corollary of a stronger result pertaining to the compositionality of the property language $L^{-}_{\forall S}$. A property language is *compositional* iff every property ϕ of a composite system $S \| T$ can be reduced to a necessary and sufficient property ϕ_T of the component S. As the property ϕ_T is required to be expressible in the property language under consideration, compositionality clearly puts a demand on its expressive power. Let L_{bad} be the property language with only one property ϕ_{nb}, expressing that no reject state can ever be reached (a simple safety property). We prove that $L^{-}_{\forall S}$ is the least expressive, compositional extension of the language L_{bad} (Thm. 5.4). This yields the desired expressive completeness result because any compositional property language that can express the property ϕ_{nb} is expressive complete with respect to reachability testing (Propn. 5.3).

The paper is organized as follows. After reviewing the variation on the model of timed automata which will be considered in this study (Sect. 2), we introduce test automata and describe how they can be used to test for properties via reachability analysis (Sect. 3). The property language studied in this paper is presented in Sect. 4. We then proceed to argue that the language $L^{-}_{\forall S}$ is testable (Sect. 4.1). Our main results are presented in Sect. 5. *Ibidem* we show that $L^{-}_{\forall S}$ is the least expressive, compositional property language that can express the aforementioned safety property ϕ_{nb}, and use this result to derive its expressive completeness with respect to reachability testing.

2 Preliminaries

Timed Labelled Transition Systems Let $\mathcal{A}$ be a finite set of *actions*, and $\mathcal{U}$ be a finite set of *urgent actions* disjoint from $\mathcal{A}$. We use Act to stand for $\mathcal{A} \cup \mathcal{U}$ and let a, b, c range over it. We assume that Act comes equipped with a mapping $\bar{\cdot} : \mathsf{Act} \to \mathsf{Act}$ such that $\overline{\overline{a}} = a$, for every $a \in \mathsf{Act}$. Moreover, we require that $\overline{a} \in \mathcal{A}$ iff $a \in \mathcal{A}$, for every action a. (Note that, since $\mathcal{A}$ and $\mathcal{U}$ are disjoint, it is also the case that $\overline{a} \in \mathcal{U}$ iff $a \in \mathcal{U}$.) We let Act_τ stand for $\mathsf{Act} \cup \{\tau\}$, where τ is a symbol not occurring in Act, and use μ to range over it. The symbol τ

will stand for an internal action of a system. Let $\mathbb{N}$ and $\mathbb{R}_{\geq 0}$ denote the sets of natural and non-negative real numbers, respectively. We use $\mathcal{D}$ to denote the set of *delay actions* $\{\epsilon(d) \mid d \in \mathbb{R}_{\geq 0}\}$, and $\mathcal{L}$ to stand for the union of Act_τ and $\mathcal{D}$. The meta-variable α will range over $\mathcal{L}$.

A *timed labelled transition system* (TLTS) is a structure $\mathcal{T} = \langle \mathcal{S}, \mathcal{L}, s^0, \longrightarrow \rangle$ where $\mathcal{S}$ is a set of *states*, $s^0 \in \mathcal{S}$ is the initial state, and $\longrightarrow \subseteq \mathcal{S} \times \mathcal{L} \times \mathcal{S}$ is a transition relation satisfying the following properties:

- (Time Determinism) for every $s, s', s'' \in \mathcal{S}$ and $d \in \mathbb{R}_{\geq 0}$, if $s \xrightarrow{\epsilon(d)} s'$ and $s \xrightarrow{\epsilon(d)} s''$, then $s' = s''$;
- (Time Additivity) for every $s, s'' \in \mathcal{S}$ and $d_1, d_2 \in \mathbb{R}_{\geq 0}$, $s \xrightarrow{\epsilon(d_1+d_2)} s''$ iff $s \xrightarrow{\epsilon(d_1)} s' \xrightarrow{\epsilon(d_2)} s''$, for some $s' \in \mathcal{S}$;
- (0-Delay) for every $s, s' \in \mathcal{S}$, $s \xrightarrow{\epsilon(0)} s'$ iff $s = s'$;
- (Forward Persistence of Urgent Actions) for every $s, s', s'' \in \mathcal{S}$, $a \in \mathcal{U}$ and $d \in \mathbb{R}_{\geq 0}$, if $s \xrightarrow{\epsilon(d)} s'$ and $s \xrightarrow{a} s''$, then there exists $\overline{s} \in \mathcal{S}$ such that $s' \xrightarrow{a} \overline{s}$;
- (Backward Persistence of Urgent Actions) for every $s, s', s'' \in \mathcal{S}$, $a \in \mathcal{U}$ and $d \in \mathbb{R}_{\geq 0}$, if $s \xrightarrow{\epsilon(d)} s'$ and $s' \xrightarrow{a} s''$, then there exists $\overline{s} \in \mathcal{S}$ such that $s \xrightarrow{a} \overline{s}$.

As usual, we write $s \xrightarrow{\alpha}$ to mean that there is some state s' such that $s \xrightarrow{\alpha} s'$, and $s \not\xrightarrow{\alpha}$ if there is no state s' such that $s \xrightarrow{\alpha} s'$.

The axioms of time determinism, time additivity and 0-delay are standard in the literature on TCCS (see, e.g., [9]). Those dealing with urgent actions are motivated by the particular kind of timed automaton model considered in verification tools like HyTech [5] and Uppaal [8].

A *delaying computation* is a sequence of transitions $s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_n} s_n$ $(n \geq 0)$ such that $\alpha_i = \tau$ or $\alpha_i \in \mathcal{D}$, for every $i \in \{1, .., n\}$. Following [9], we now proceed to define versions of the transition relations that abstract from the internal evolution of states as follows:

$$
\begin{aligned}
s \overset{a}{\Longrightarrow} s' \quad &\text{iff} \quad \exists s''.\quad s \xrightarrow{\tau}{}^{*} s'' \xrightarrow{a} s' \\
s \overset{\epsilon(d)}{\Longrightarrow} s' \quad &\text{iff} \quad \text{there exists a delaying computation} \\
&\qquad s = s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_n} s_n = s' \text{ with } d = \sum \{d_i \mid \alpha_i = \epsilon(d_i)\}
\end{aligned}
$$

By convention, if the set $\{d_i \mid \alpha_i = \epsilon(d_i)\}$ is empty, then $\sum\{d_i \mid \alpha_i = \epsilon(d_i)\}$ is 0. We define a collection of transition relations parameterized by a set of urgent actions S as follows:

$$s \xrightarrow{\epsilon(d)}_S s' \quad \text{iff} \quad s \xrightarrow{\epsilon(d)} s' \text{ and } \forall d' \in [0,d[, a \in S, s' \in \mathcal{S}.\ s \xrightarrow{\epsilon(d')} s' \text{ implies } s' \not\xrightarrow{a}$$

$$s \overset{\epsilon(d)}{\Longrightarrow}_S s' \quad \text{iff} \quad \begin{array}{l} \text{there exists a delaying computation} \\ s = s_0 \xrightarrow{\alpha_1}_S s_1 \xrightarrow{\alpha_2}_S \ldots \xrightarrow{\alpha_n}_S s_n = s' \text{ with} \\ d = \sum \{d_i \mid \alpha_i = \epsilon(d_i)\} \end{array}$$

where the relation $\xrightarrow{\tau}_S$ coincides with $\xrightarrow{\tau}$. Intuitively, $s \xrightarrow{\epsilon(d)}_S s'$ holds if s can delay d units of time, and no action in the set S becomes enabled before time d during this delay activity. Note that, since the set S only contains urgent actions, this amounts to requiring that either $d = 0$ or $s \not\xrightarrow{a}$, for every $a \in S$. Similarly, $s \overset{\epsilon(d)}{\Longrightarrow}_S s'$ holds if there exists a delaying computation of duration d from state s whose delay transitions with positive duration occur only in states in which none of the urgent actions in S are enabled.

Definition 2.1 (Operations on TLTSs).

- Let $\mathcal{T}_i = \langle \mathcal{S}_i, \mathcal{L}, s_i^0, \longrightarrow_i \rangle$ $(i \in \{1,2\})$ be two TLTSs. The parallel composition of $\mathcal{T}_1$ and $\mathcal{T}_2$ is the TLTS $\mathcal{T}_1 \parallel \mathcal{T}_2 = \langle \mathcal{S}_1 \times \mathcal{S}_2, \mathcal{L}, (s_1^0, s_2^0), \longrightarrow \rangle$, where the transition relation $\longrightarrow$ is defined by the rules in below:

$$(1)\quad \frac{s_1 \xrightarrow{\mu}_1 s_1'}{s_1 \| s_2 \xrightarrow{\mu} s_1' \| s_2} \qquad\qquad (2)\quad \frac{s_2 \xrightarrow{\mu}_2 s_2'}{s_1 \| s_2 \xrightarrow{\mu} s_1 \| s_2'}$$

$$(3)\quad \frac{s_1 \xrightarrow{a}_1 s_1' \qquad s_2 \xrightarrow{\overline{a}}_2 s_2'}{s_1 \| s_2 \xrightarrow{\tau} s_1' \| s_2'}$$

$$(4)\quad \frac{s_1 \xrightarrow{\epsilon(d)}_1 s_1' \qquad s_2 \xrightarrow{\epsilon(d)}_2 s_2'}{s_1 \| s_2 \xrightarrow{\epsilon(d)} s_1' \| s_2'} \qquad \begin{array}{l} d = 0 \text{ or } \forall a \in \mathcal{U}. \\ \neg(s_1 \xrightarrow{a}_1 \ \wedge\ s_2 \xrightarrow{\overline{a}}_2) \end{array}$$

  where s_i, s_i' are states of $\mathcal{T}_i$ $(i \in \{1,2\})$, $\mu \in \mathsf{Act}_\tau$, $a, \overline{a} \in \mathsf{Act}$ and $d \in \mathbb{R}_{\geq 0}$. In the above rules, and in the remainder of the paper, we use the more suggestive notation $s \parallel s'$ in lieu of (s, s').
- Let $\mathcal{T} = \langle \mathcal{S}, \mathcal{L}, s^0, \rightarrow \rangle$ be a TLTS and let $L \subseteq \mathsf{Act}$ be a set of actions. The restriction of $\mathcal{T}$ over L is the TLTS $\mathcal{T} \backslash L = \langle \mathcal{S} \backslash L, \mathcal{L}, s^0 \backslash L, \leadsto \rangle$, where $\mathcal{S} \backslash L = \{s \backslash L \mid s \in \mathcal{S}\}$ and the transition relation $\leadsto$ is defined by the rules below:

$$(1)\quad \frac{s \xrightarrow{\tau} s'}{s \backslash L \overset{\tau}{\leadsto} s' \backslash L} \qquad\qquad (2)\quad \frac{s \xrightarrow{\epsilon(d)} s'}{s \backslash L \overset{\epsilon(d)}{\leadsto} s' \backslash L}$$

$$(3)\quad \frac{s \xrightarrow{a} s'}{s \backslash L \overset{a}{\leadsto} s' \backslash L} \qquad\qquad a, \overline{a} \notin L$$

  where s, s' are states of $\mathcal{T}$, $L \subseteq \mathsf{Act}$, $a \in \mathsf{Act}$, and $d \in \mathbb{R}_{\geq 0}$.

The reader familiar with TCCS [9] may have noticed that the above definition of parallel composition has strong similarities with that of TCCS parallel composition — the only difference being that in TCCS *all* actions are urgent. This yields precisely the parallel composition operator used in UPPAAL [8].

Timed Automata Let C be a set of clocks. We use $\mathcal{B}(C)$ to denote the set of boolean expressions over atomic formulae of the form $x \sim p$ and $x - y \sim p$, with $x, y \in C$, $p \in \mathbb{N}$, and $\sim \in \{<, >, =\}$. Expressions in $\mathcal{B}(C)$ are interpreted over the collection of time assignments. A *time assignment*, or *valuation*, v for C is a function from C to $\mathbb{R}_{\geq 0}$. Given a condition $g \in \mathcal{B}(C)$ and a time assignment v, the boolean value $g(v)$ describes whether g is satisfied by v or not. (Note that $\mathcal{B}(C)$ is closed under negation.) For every time assignment v and $d \in \mathbb{R}_{\geq 0}$, we use $v + d$ to denote the time assignment which maps each clock $x \in C$ to the value $v(x) + d$. Two assignments u and v are said to agree on the set of clocks C' iff they assign the same real number to every clock in C'. For every subset C' of clocks, $[C' \to 0]v$ denotes the assignment for C which maps each clock in C' to the value 0 and agrees with v over $C \backslash C'$. For an assignment u and a subset C' of C, we write $u \upharpoonright C'$ for the restriction of u to the set of clocks C'. Given two disjoint sets of clocks C_1, C_2, and two valuations v_1, v_2 for the clocks of C_1 and C_2 respectively, $v_1 : v_2$ denotes the valuation for the clocks of $C_1 \cup C_2$ such that $(v_1 : v_2)(x) = v_1(x)$ iff $x \in C_1$ and $(v_1 : v_2)(x) = v_2(x)$ iff $x \in C_2$.

The notion of timed automaton we use in this paper is a variation on the original one introduced by Alur and Dill [3], and underlies that used in, e.g., HyTech [5] and UPPAAL [8].

Definition 2.2. A *timed automaton* is a tuple $A = \langle \mathsf{Act}_\tau, N, n_0, C, E \rangle$ where N is a finite set of *nodes*, n_0 is the *initial node*, C is a finite set of *clocks*, and $E \subseteq N \times N \times \mathsf{Act}_\tau \times 2^C \times \mathcal{B}(C)$ is a set of *edges*. The tuple $e = \langle n, n_e, \mu, r_e, g_e \rangle \in E$ stands for an edge from node n to node n_e (the *target* of e) with action μ, where r_e denotes the set of clocks to be reset to 0 and g_e is the enabling condition (or *guard*) over the clocks of A. All the timed automata we shall consider in this paper will satisfy the following constraint:

- (URGENCY) if $\langle n, n_e, \mu, r_e, g_e \rangle \in E$ and $\mu \in \mathcal{U}$, then g_e is a tautology, i.e., g_e is satisfied by every valuation for the clocks in C.

In what follows, we shall assume that the clocks used in timed automata come from a fixed, countably infinite collection of clocks C_A.

The timed automaton depicted in Figure 1 has five nodes labelled n_0 to n_4, one clock x, actions $a \in \mathcal{U}$ and $b \in \mathcal{A}$, and four edges. The edge from node n_1 to node n_2, for example, is guarded by $x \geq 0$, is labelled with the urgent action a and resets clock x. Note that the guards of edges labelled with the urgent action a are tautologies. A *state* of a timed automaton A is a pair (n, v) where n is a node of A and v is a time assignment for C. The operational semantics of a timed automaton A is given by the TLTS $\mathcal{T}_A = \langle \mathcal{S}, \mathcal{L}, (n_0, [C \to 0]), \longrightarrow \rangle$, where $\mathcal{S}$ is the set of states of A, and $\longrightarrow$ is the transition relation defined as follows ($\mu \in \mathsf{Act}_\tau$, $\epsilon(d) \in \mathcal{D}$):

$$(n, v) \xrightarrow{\mu} (n', v') \text{ iff } \exists e = \langle n, n', \mu, r_e, g_e \rangle \in E.\ g_e(v) \wedge v' = [r_e \to 0]v$$

$$(n, v) \xrightarrow{\epsilon(d)} (n', v') \text{ iff } n = n' \text{ and } v' = v + d$$

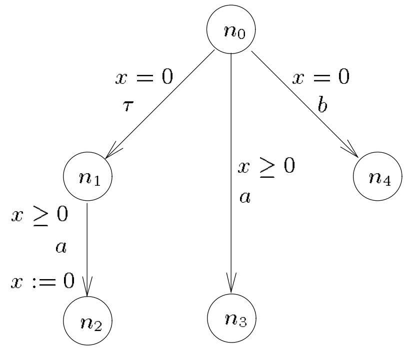


Figure 1: Timed automaton A ($a \in \mathcal{U}$ and $b \in \mathcal{A}$)

3 Testing Automata

As mentioned in Sect. 1, the main aim of this paper is to present a complete characterization of the class of properties of (networks of) timed automata for which model checking can be reduced to reachability analysis. In this section we take the first steps towards the definition of *model checking via (reachability) testing* by defining *testing*. Informally, testing involves the parallel composition of the tested automaton with a *test automaton*. The testing process then consists in performing reachability analysis in the composed system restricted over all non internal actions. We say that the tested automaton fails the test if a special reject state of the test automaton is reachable in the parallel composition (restricted over all non internal actions) from their initial configurations, and passes otherwise. The formal definition of testing then involves the definition of what a test automaton is, how the parallel composition is performed and when the test has failed or succeeded. We now proceed to make these notions precise.

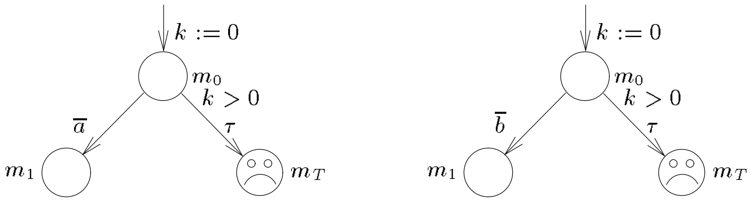
Definition 3.1. A *test automaton* is a tuple $T = \langle \mathsf{Act}_\tau, N, N_T, n_0, C, C_0, E \rangle$ where Act_τ, N, n_0, C, and E are as in Definition 2.2, $N_T \subseteq N$ is the set of *reject nodes*, and $C_0 \subseteq C$ is the set of clocks whose value must be 0 at the beginning of every run of the automaton.

An *initial valuation* for T is any valuation for the set of clocks C that assigns the value 0 to every clock in C_0. An *initial state* of T is any state (n_0, u_0) of T with u_0 an initial valuation.

In what follows, we shall assume that the clocks used in test automata come from a fixed, countably infinite collection of clocks C_T disjoint from C_A.

Definition 3.2. Let $\mathcal{T}$ be a TLTS and let T be a test automaton. We say that a node n of T is reachable from a state $(s_1 \parallel s_2) \backslash \mathsf{Act}$ of $(\mathcal{T} \parallel \mathcal{T}_T) \backslash \mathsf{Act}$ iff there is a delaying computation leading from $(s_1 \parallel s_2) \backslash \mathsf{Act}$ to a state whose $\mathcal{T}_T$ component is of the form (n, u).

A state s of $\mathcal{T}$ *fails the test* T from the initial state (n_0, u_0) iff a reject node of T is reachable in $(\mathcal{T} \parallel \mathcal{T}_T) \backslash \mathsf{Act}$ from the state $(s \parallel (n_0, u_0)) \backslash \mathsf{Act}$. Otherwise, we say that s *passes the test* T from the initial state (n_0, u_0).



(a) (b)

Figure 2: The test automata T_a and T_b

Example 3.3. Consider the timed automaton A of Figure 1 and the test automaton T_b ($b \in \mathcal{U}$) of Figure 2(b), where we label the arrow coming into the initial node m_0 of T_b with the assignment $k := 0$ to denote the fact that clock k is contained in C_0. (This convention will be used throughout the paper.) The reject node m_T of the test automaton is reachable from the initial state of $(A \parallel T_b) \backslash \mathsf{Act}$ as follows. First the automaton A can execute the τ-transition and go to node n_1, thus preempting the possibility of synchronizing on channel b with T_b. Next both automata can let a positive amount of time pass, thus enabling the τ-transition from node m_0 in T_b and making m_T reachable. In this case we say that A fails the test. If we test A using the automaton T_a ($a \in \mathcal{U}$) of Figure 2(a), then, in all cases, A and T_a must synchronize on a and, since a is urgent, no positive initial delay is possible. It follows that the reject node m_T of T_a is unreachable, and A passes the test.

4 Property Languages

In our previous study [2] we considered SBLL, a dense-time property language with clocks suitable for the specification of safety and bounded liveness properties of TLTSs. For the sake of clarity in the subsequent discussion, we now recall the syntax of the language SBLL — modified to take into account the current distinction between urgent and non-urgent actions. The interested reader is referred to [2] for more information.

Definition 4.1 (The Property Language SBLL**).** Let K be a countably infinite set of clocks, disjoint from C_A and including C_T. We use `fail` to denote an action symbol not contained in Act. The set SBLL of formulae over K is generated by the following grammar:

$$\begin{array}{lcl} \varphi & ::= & \mathtt{ff} \ \mid\ \varphi_1 \wedge \varphi_2 \ \mid\ g \vee \varphi \ \mid\ \forall\!\!\forall \varphi \ \mid \\ & & [a]\varphi \ \mid\ \langle a \rangle \mathtt{tt}\ \ (a \in \mathcal{U}) \ \mid\ x\ \underline{\mathtt{in}}\ \varphi \ \mid\ X \ \mid\ \mathtt{max}(X, \varphi) \\ g & ::= & x \sim p \ \mid\ x - y \sim p \end{array}$$

where $a \in \mathsf{Act} \cup \{\mathtt{fail}\}$, $x, y \in K$, $p \in \mathbb{N}$, $\sim \in \{<, >, =\}$, X is a formula variable and $\mathtt{max}(X, \varphi)$ stands for the maximal solution of the recursion equation $X = \varphi$.

Following [7], the formulae in SBLL were interpreted in [2] over extended states of TLTSs, i.e., over pairs of the form $\langle s, v \rangle$, where s is a state of a TLTS and v is a valuation for the clocks in K. For the sake of clarity in the presentation, we recall that the satisfaction relation for SBLL is the largest relation satisfying the relevant implications in Table 1 below and

$$\langle s, u \rangle \models \forall\!\!\forall \varphi \;\Rightarrow\; \forall d \in \mathbb{R}_{\geq 0}, \forall s'.\ s \overset{\epsilon(d)}{\Longrightarrow} s' \text{ implies } \langle s', u + d \rangle \models \varphi \ .$$

Our main result in *op. cit.* was that the property language SBLL is testable over states of timed automata, in the sense that, for every formula $\varphi \in$ SBLL, we can construct a test automaton T_φ such that every extended state $\langle s, u \rangle$ of a timed automaton satisfies φ iff it passes the test T_φ, in the sense of Defn. 3.2. It is now natural to wonder whether every property φ that is testable in this fashion can be expressed in the property language SBLL. This amounts to asking whether every test automaton T is expressible in the language SBLL, in the sense that there exists a formula ψ_T of SBLL such that every timed automaton A passes the test T if and only if A satisfies ψ_T.

The starting point of our current investigation is the realization that test automata have a greater expressive power than the specification language SBLL. As an example, consider the test automaton T depicted in Fig. 3, where a is an urgent action. It can be shown that the property tested by the automaton in

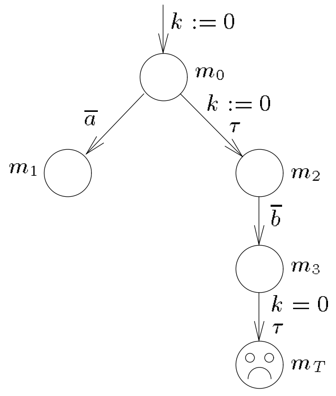


Figure 3: A test automaton that cannot be expressed in SBLL ($a \in \mathcal{U}$)

Fig. 3 is *not* expressible in the language SBLL. Intuitively, the property that is tested by the automaton in Fig. 3 requires that, by delaying without enabling an a action in the process, a state can only evolve to one in which it can*not* perform the action b.

The kind of test automaton depicted in Fig. 3 suggests an enrichment of the property language SBLL in which the delay construct $\forall\!\!\forall$ is parameterized by a set of urgent actions, whose elements should not become enabled as a state delays. It is perhaps surprising that, as we shall show in the sequel (cf. Thm. 5.5), this simple extension of SBLL yields a property language that is expressive complete with respect to the collection of reachability properties expressible by means of test automata, in the sense of Defn. 3.2.

The property language we study here is an extension of the one considered in [2] (cf. Defn. 4.1), and is closely related to the modal logic L_ν presented in [7], and further investigated in [6].

Definition 4.2. The property language $L_{\forall S}$ consists of the formulae over K generated by the grammar obtained from the one in Defn. 4.1 by replacing constructs of the form $\forall\!\!\!\forall\varphi$ with $\forall\!\!\!\forall_S\varphi$, where S is a collection of urgent actions. We use $L^-_{\forall S}$ to stand for the collection of formulae in $L_{\forall S}$ that do not contain occurrences of the basic propositions $\langle a\rangle\mathtt{tt}$.

We write g in lieu of $g \vee \mathtt{ff}$, and clocks(φ) for the collection of clocks occurring in the formula φ. We use the standard definition of closed formulae. In the remainder of this paper, every formula will be closed.

Given a TLTS $\mathcal{T} = \langle S, \mathcal{L}, s^0, \longrightarrow\rangle$, we interpret, as usual, the closed formulae in $L_{\forall S}$ over extended states. We recall that an *extended state* is a pair $\langle s, u\rangle$ where s is a state of $\mathcal{T}$ and u is a time assignment for the formula clocks in K. The satisfaction relation $\models$ is the largest relation included in $\mathcal{S} \times L_{\forall S}$ satisfying the implications in Table 1. We refer the reader to [2] for a discussion of the definition of $\models$. Note that, since `fail` is not contained in Act, every extended state of a TLTS trivially satisfies formulae of the form $[\mathtt{fail}]\phi$. The role played by these formulae in the developments of this paper will become clear in Sect. 5.

4.1 Testing $L^-_{\forall S}$

In Sect. 3 we have seen how we can perform tests on states of TLTSs. We now aim at using test automata to determine whether a given state of a TLTS satisfies a formula in $L^-_{\forall S}$.

Definition 4.3 (Testing Properties). Let φ be a formula in $L_{\forall S}$, and consider a test automaton $T_\varphi = \langle \mathsf{Act}_\tau, N, N_T, m_0, C, C_0, E\rangle$. For every extended state $\langle s, u\rangle$ of a TLTS $\mathcal{T}$, we say that $\langle s, u\rangle$ passes the test T_φ iff no reject node of T_φ is reachable from the state $(s \parallel (m_0, [C_0 \to 0](u \upharpoonright C)))\backslash\mathsf{Act}$.

The test automaton T_φ *tests* for the formula φ (and we say that φ is *testable*) iff the following holds: for every TLTS $\mathcal{T}$ and every extended state $\langle s, u\rangle$ of $\mathcal{T}$,

$$\langle s, u\rangle \models \varphi \text{ iff } \langle s, u\rangle \text{ passes the test } T_\varphi \ . \tag{1}$$

If (1) holds for arbitrary states of timed automata then we say that the test automaton T_φ *tests* for the formula φ (and that φ is *testable*) over states of timed automata.

Adapting constructions first developed in [2], we can now prove that:

Theorem 4.4. *Every formula in $L^-_{\forall S}$ is testable, and every formula in $L_{\forall S}$ is testable over states of timed automata.*

We remark here that the property languages SBLL and $L_{\forall S}$ are *not* testable because there is no test automaton for the formula $\langle a\rangle\mathtt{tt}$. On the other hand, the languages $L_{\forall S}$ and $L^-_{\forall S}$ are equally expressive over states of timed automata. We refer the interested reader to the full version of this work [1] for more details.

$\langle s,u\rangle \models \mathtt{ff}$	$\Rightarrow$	$false$
$\langle s,u\rangle \models \varphi_1 \wedge \varphi_2$	$\Rightarrow$	$\forall s'.\ s \xrightarrow{\tau}{}^{*} s'$ implies $\langle s',u\rangle \models \varphi_1$ and $\langle s',u\rangle \models \varphi_2$
$\langle s,u\rangle \models g \vee \varphi$	$\Rightarrow$	$\forall s'.\ s \xrightarrow{\tau}{}^{*} s'$ implies $g(u)$ or $\langle s',u\rangle \models \varphi$
$\langle s,u\rangle \models [a]\varphi$	$\Rightarrow$	$\forall s'.\ s \stackrel{a}{\Longrightarrow} s'$ implies $\langle s',u\rangle \models \varphi$
$\langle s,u\rangle \models \langle a\rangle\mathtt{tt}$	$\Rightarrow$	$\forall s'.\ s \xrightarrow{\tau}{}^{*} s'$ implies $s' \xrightarrow{a} s''$ for some s''
$\langle s,u\rangle \models \forall\!\!\!\forall_S\,\varphi$	$\Rightarrow$	$\forall d \in \mathbb{R}_{\geq 0}\, \forall s'.\ \ s \stackrel{\epsilon(d)}{\Longrightarrow}_S s'$ implies $\langle s',u+d\rangle \models \varphi$
$\langle s,u\rangle \models x\ \underline{\mathtt{in}}\ \varphi$	$\Rightarrow$	$\forall s'.\ \ s \xrightarrow{\tau}{}^{*} s'$ implies $\langle s',[x \to 0]u\rangle \models \varphi$
$\langle s,u\rangle \models \mathtt{max}(X,\varphi)$	$\Rightarrow$	$\forall s'.\ \ s \xrightarrow{\tau}{}^{*} s'$ implies $\langle s',u\rangle \models \varphi\{\mathtt{max}(X,\varphi)/X\}$

Table 1: Satisfaction implications

5 Compositionality and Expressive Completeness

We have previously shown that every property φ which can be expressed in the language $L_{\forall S}$ (and, *a fortiori*, in SBLL) is testable over states of timed automata, and that $L^-_{\forall S}$ is testable over states of TLTSs, in the sense of Defn. 4.3. We now address the problem of the expressive completeness of these property languages with respect to test automata and (reachability) testing. More precisely, we study whether all properties that are testable over TLTSs can be expressed in the property languages SBLL and $L^-_{\forall S}$—in the sense that, for every test automaton T, there exists a formula ψ_T such that every extended state of a TLTS passes the test T if and only if it satisfies ψ_T. Indeed, we have already enough information to claim that the language SBLL is strictly less expressive than the formalism of test automata. In fact, the automaton depicted in Fig. 3 is nothing but a test automaton for the formula $\forall\!\!\!\forall_{\{a\}}[b]\mathtt{ff}$, which cannot be expressed in SBLL. Our aim in this section is to argue that, unlike SBLL, the language $L^-_{\forall S}$ *is* expressive complete, in the sense that every test automaton T may be expressed as a property in the language $L^-_{\forall S}$ in the precise technical sense outlined above. In the proof of this expressive completeness result, we shall follow an indirect approach by focusing on the compositionality of a property language $\mathbb{L}$ with respect to test automata and the parallel composition operator $\|$. As we shall see (cf. Propn. 5.3), if a property language $\mathbb{L}$ is compositional with respect to timed automata and $\|$ (cf. Defn. 5.2) then it is complete with respect to test automata and reachability testing (cf. Defn. 5.1). We begin with some preliminary definitions, introducing the key concepts of compositionality and (expressive) completeness.

Definition 5.1 (Expressive completeness). A property language $\mathbb{L}$ over the set of clocks K is *(expressive) complete* (with respect to test automata and testing) if for every test automaton T there exists a formula $\varphi_T \in \mathbb{L}$ such that, for every extended state $\langle s,u\rangle$ of a TLTS, $\langle s,u\rangle \models \varphi_T$ iff $\langle s,u\rangle$ passes the test T.

Compositionality, on the other hand, is formally defined as follows [6]:

Definition 5.2 (Compositionality). A property language $\mathbb{L}$ over the set of clocks K is *compositional* (with respect to test automata and $\|$) if, for every $\varphi \in \mathbb{L}$ and every test automaton $T = \langle \mathsf{Act}_\tau, N, N_T, n_0, C, C_0, E\rangle$ (with C disjoint from clocks(φ)), there exists a formula $\varphi/T \in \mathbb{L}$ over the set of clocks $C \cup \text{clocks}(\varphi)$ such that, for every state s of a TLTS and every valuation u for K,

$$\langle s \parallel (n_0, [C_0 \to 0](u \restriction C)), u\rangle \models \varphi \;\Leftrightarrow\; \langle s, [C_0 \to 0]u\rangle \models \varphi/T \ .$$

Our interest in compositionality stems from the following result that links it to the notion of completeness. In the sequel, we use $\mathbb{L}_{bad}$ to denote the property language that only consists of the formula $\forall\!\!\!\forall_\emptyset[\mathtt{fail}]\mathtt{ff}$, where `fail` is a fresh action not contained in Act.

Proposition 5.3. *Let $\mathbb{L}$ be a property language (over a set of clocks K) that includes $\mathbb{L}_{bad}$. Suppose that $\mathbb{L}$ is compositional with respect to test automata and the parallel composition operator $\|$. Then $\mathbb{L}$ is complete with respect to test automata and testing.*

Since $L_{\forall S}^-$ is an extension of $\mathbb{L}_{bad}$, in light of the above proposition an approach to proving that it is expressive complete is to establish that it is compositional with respect to test automata and $\|$. This is the import of the following stronger result:

Theorem 5.4. *The property language $L_{\forall S}^-$ is the least expressive extension of $\mathbb{L}_{bad}$ that is compositional with respect to test automata and $\|$.*

In light of Propn. 5.3, we may finally obtain that:

Theorem 5.5. *The property language $L_{\forall S}^-$ is complete with respect to test automata and testing.*

For example, the properties tested by the test automata in Figs. 2(a) and 3 may be expressed in $L_{\forall S}^-$ as $k\ \underline{\mathtt{in}}\ \forall\!\!\!\forall_{\{a\}}(k = 0)$ and $\forall\!\!\!\forall_{\{a\}}[b]\mathtt{ff}$, respectively.

It is interesting to remark here that the property language $L_{\forall S}^-$ is testable also over timed automata with invariants and committed nodes, which are precisely those used in UPPAAL. Moreover, Thm. 5.5 also holds for the model of timed automata with invariants. We refer the interested reader to [4] for details on these results.

References

1. L. Aceto, P. Bouyer, A. Burgueño, and K. G. Larsen, *The power of reachability testing for timed automata*, 1998. Forthcoming paper.
2. L. Aceto, A. Burgueño, and K. G. Larsen, *Model checking via reachability testing for timed automata*, in Proceedings of TACAS '98, Lisbon, B. Steffen, ed., vol. 1384 of Lecture Notes in Computer Science, Springer-Verlag, 1998, pp. 263–280. Also available as BRICS Report RS–97–29, Aalborg University, November, 1997.
3. R. Alur and D. Dill, *A theory of timed automata*, Theoretical Computer Science, 126 (1994), pp. 183–235.

4. A. Burgueño, *Model-checking via Testing and Parametric Analysis of Timed Systems*, PhD thesis, École Nationale Supérieure de l'Aéronautique et de l'Éspace, Toulouse, France, June 1998.
5. T. A. Henzinger, P.-H. Ho, and H. Wong-Toi, *HyTech: the next generation*, in Proc. of the 16th Real-time Systems Symposium, RTSS'95, IEEE Computer Society press, 1995.
6. F. Laroussinie and K. G. Larsen, *Compositional model checking of real time systems*, in Proc. of the 6th. International Conference on Concurrency Theory, CONCUR'95, I. Lee and S. Smolka, eds., vol. 962 of Lecture Notes in Computer Science, Philadelphia, PA, USA, August 21 - 24 1995, Springer-Verlag.
7. F. Laroussinie, K. G. Larsen, and C. Weise, *From timed automata to logic - and back*, in Proc. of the 20th. International Symposium on Mathematical Foundations of Computer Science, MFCS'95, J. Wiedermann and P. Hájek, eds., vol. 969 of Lecture Notes in Computer Science, Prague, Czech Republic, August 28 - September 1 1995, Springer-Verlag, pp. 529–539.
8. K. G. Larsen, P. Pettersson, and Y. Wang, Uppaal *in a nutshell*, Software Tools for Technology Transfer, 1 (1997), pp. 134–152.
9. Y. Wang, *Real-time behaviour of asynchronous agents*, in Proc. of the Conference on Theories of Concurrency: Unification and Extension, CONCUR'90, J. Baeten and J. Klop, eds., vol. 458 of Lecture Notes in Computer Science, Amsterdam, The Netherlands, August 27–30 1990, Springer-Verlag, pp. 502–520.