Testing Timed Automata*

Jan Springintveld[1†] Frits Vaandrager[1] Pedro R. D'Argenio[2]

[1] Computing Science Institute
University of Nijmegen
P.O. Box 9010, 6500 GL Nijmegen
The Netherlands
{jans,fvaan}@cs.kun.nl

[2] Department of Computer Science
University of Twente
P.O. Box 217, 7500 AE Enschede
The Netherlands
dargenio@cs.utwente.nl

Abstract

We present a generalization of the classical theory of testing for Mealy machines to a setting of dense real-time systems. A model of *timed I/O automata* is introduced, inspired by the timed automaton model of Alur and Dill, together with a notion of test sequence for this model. Our main contribution is a test suite derivation algorithm for black-box conformance testing of timed I/O automata. It is based on reducing the original specification into an appropriate discrete model which contains enough information to completely represent the dense time model. The test suite is then obtained from the discrete model.

Although the method results in a test suite of high exponential size and cannot be claimed to be of practical value, it gives the first algorithm that yields a finite and complete set of tests for dense real-time systems.

Keywords and Phrases: (black-box) conformance testing, real-time systems, timed automata, I/O automata, bisimulation.
AMS Subject Classification (1991): 68M15, 68Q05, 68Q68.
CR Subject Classification (1991): B.4.5, C3, D.2.5, F.1.1.

1 Introduction

It is widely recognized that testing is an essential component of the life cycle of computer systems [Som96]. Black-box testing is the approach to testing which relies on the specification of the system which is being tested to derive test cases. It permits to define the notion of a *conformance* relation linking the system to the specification, and the notion of a *verdict* associated to the application of a test case. In general, testing can only demonstrate the presence, not the absence, of system faults. However, if we have reason to believe that the system behaves according to some (unknown) formal model within a given (known) finite class of formal models, then it is in some cases possible to generate a finite and *complete* set of test cases (the *test suite*) and use it to demonstrate the absence of system faults under the assumption that our belief is correct. This assumption is usually referred to as the test hypothesis [Tre92]. The last two decades have witnessed a lot of research activity in the area of (black-box) conformance testing. Especially for the class of finite state models, many algorithms to derive test suites have been devised, which have been used successfully for the validation of hardware circuits and communication protocols

*Research supported by the Netherlands Organization for Scientific Research (NWO) under contract SION 612-33-006 and by the HCM network EXPRESS.

†Current affiliation: Philips Research Laboratories Eindhoven, Prof. Holstlaan 4, 5656 AA - Eindhoven, The Netherlands, springtv@natlab.research.philips.com.

[Koh78, Cho78, CVI89, ADLU91, Hol91]. So far, however, little work has been done to incorporate timing aspects [CL97, MMM95, CG98]. An important reason for this has no doubt been the lack of a suitable model for timed systems.

Recently, Alur and Dill [AD94] have proposed the model of *timed automata*, which is an extension of the finite automaton model with clock variables and simple constraints over clocks and states. The timed automata model and its variants have been used quite successfully for verification purposes and form the basis for several model-checking tools [AK96, LPY97, DOTY96, HH95]. Although the algorithms involved are theoretically of high complexity, analysis of non-trivial timed systems turns out to be feasible, as is witnessed by several case studies [BGK$^+$96, DKRT97, DY95, HWT95].

This article is a first step towards a theory of testing for timed automata. We propose a model of *timed I/O automata*, which borrows ideas from both Alur and Dill's model and from the timed I/O automata of Lynch et al. [SGSL98]. Apart from supporting the automatic generation of timed tests, our model allows a loose coupling of inputs and outputs, unlike the usual Mealy style finite state machines where inputs and outputs occur simultaneously in a single transition. A similar (even more flexible) modeling of input and outputs is presented in [Tre96b, Tre96a, FJJV96], but this approach does not deal with time.

We provide a method to derive a complete test suite in the style of well-known finite state machine based methods (see, e.g., [Cho78, CVI89, ADLU91]). The main problem involved is that in general the state space of a timed automaton is (uncountably) infinite. To obtain a finite test suite, a discretization of the state space is required which is still sufficiently refined to detect all possible errors.

To the best of our knowledge, this article proposes the first algorithm that (albeit under some strong assumptions) yields a finite and complete test suite for (dense) real-time systems. Even though the method results in a test suite of high exponential size and cannot be claimed to be of practical value, we believe that the concepts and techniques developed in this article will allow for more practical algorithms. We have not yet been able to provide lower bounds to the size of complete test suites for timed systems. We do give an example that shows that for timed systems very small time steps have to be considered, resulting in large test suites. In any case, we will sketch several possible optimizations to substantiate the belief that our approach can be turned into practice, at least in more restrictive settings. Furthermore, we hope that our approach may also support incomplete but practically useful methods for testing timed systems such as in [CL97, MMM95].

The organization of this article is as follows. In Section 2, we present the model of timed I/O automata. This model requires a new, timed notion of distinguishing sequence, which is the subject of Section 3. In Section 4 we present the basic definitions and theorems for the discretization of state spaces. These are employed in Section 5, where we present an algorithm for test generation and a proof of its correctness. Finally, in Section 6 we discuss several options to obtain more practical algorithms. An appendix lists some notational conventions.

2 Timed I/O Automata

In this section we present the model of timed I/O automata, which borrows ideas from both Alur and Dill's model [AD94] and from the timed I/O automata of Lynch et al. [SGSL98]. Our model is defined in several steps.

After recalling some basic definitions, mainly to fix notation, we present the bounded time domain automata model from [SV96], which is a variant of the model of Alur and Dill [AD94]. Roughly speaking, a *bounded time domain automaton* is a finite (untimed) automaton together with a timing annotation. This timing annotation extends the automaton with a finite set of clocks and functions that allow one to express, for each transition, under what timing conditions the transition may be taken, what the updated clock values will be, and under what timing conditions one may idle in each state. Thereafter, *timed I/O automata* are defined as bounded time domain automata together with a partitioning of the set of actions into input and output actions. We impose certain restrictions on the model to ensure 'testability' of the model. The definitions are illustrated in Example 2.7. Example 2.8 shows how timed I/O automata can be viewed naturally as a generalization of the classical Mealy machines.

2.1 Preliminaries

Let R denote the set of reals, $\mathsf{R}^{\geq 0}$ the set of nonnegative reals, $\mathsf{R}^{>0}$ the set of positive reals, and R^{∞} the set of reals together with the single element ∞. We extend the standard partial ordering $\leq$ and addition operator $+$ over R to R^{∞} in the usual way: for every $t \in \mathsf{R}^{\infty}$, $t \leq \infty$ and $t + \infty = \infty + t = \infty$. Let Z denote the set of integers, Z^{∞} the set $\mathsf{Z} \cup \{\infty\}$, and N the set of nonnegative integers. For $t \in \mathsf{R}$, $\lfloor t \rfloor$ denotes the largest number in Z that is not greater than t, and $\lceil t \rceil$ denotes the smallest number in Z that is not smaller than t. With $\mathit{fract}(t)$ we denote the fractional part of t (so $\mathit{fract}(t) = t - \lfloor t \rfloor$).

Concatenation of a finite sequence with a finite or infinite sequence is denoted by juxtaposition; ϵ denotes the empty sequence and the sequence containing a single element a is simply denoted a. If σ is a nonempty sequence then $\mathit{first}(\sigma)$ returns the first element of σ. Moreover, if σ is finite, then $\mathit{last}(\sigma)$ returns the last element of σ. If σ is a sequence and X is a set, then $\sigma\lceil X$ denotes the sequence obtained by projecting σ on X. If V is a set of finite sequences, W a set of sequences, and σ a finite sequence, then $\sigma\, W = \{\sigma\,\tau \mid \tau \in W\}$ and $V\, W = \bigcup_{\sigma \in V} \sigma\, W$. For X a set of symbols, we define $X^0 = \{\epsilon\}$ and, for $i > 0$, $X^i = X^{i-1} \cup X\, X^{i-1}$. As usual, $X^* = \bigcup_{i \in \mathsf{N}} X^i$.

2.2 Labeled Transition Systems

For technical reasons, our definition of a labeled transition system is slightly different from the standard one in which a transition is a triple of a state, an action and a state. According to our definition there can be multiple transitions with the same action label between any given pair of states.

Definition 2.1. A *labeled transition system (LTS)* is a rooted, edge-labeled multigraph. Formally, an LTS is a structure $\mathcal{A} = (Q, E, \Sigma, \mathsf{src}, \mathsf{act}, \mathsf{trg}, q^0)$, where Q is a set of *states*, E a set of *transitions*, Σ a set of *actions*, functions $\mathsf{src} : E \to Q$, $\mathsf{act} : E \to \Sigma$ and $\mathsf{trg} : E \to Q$ associate to each transition a *source*, *action* and *target*, respectively, and $q^0 \in Q$ is the *initial state*. We write $Q_{\mathcal{A}}$, $E_{\mathcal{A}}$, etc., for the components of an LTS $\mathcal{A}$, but often omit subscripts when they are clear from the context. Also, we write $\delta : q \xrightarrow{a} q'$ if δ is a transition with $\mathsf{src}(\delta) = q$, $\mathsf{act}(\delta) = a$ and $\mathsf{trg}(\delta) = q'$. With $q \xrightarrow{a} q'$ we denote that $\delta : q \xrightarrow{a} q'$ for some δ. □

An LTS $\mathcal{A}$ is *lean* if each transition is fully determined by its source, action and target, i.e.,

$$\mathsf{src}(\delta) = \mathsf{src}(\delta') \wedge \mathsf{act}(\delta) = \mathsf{act}(\delta') \wedge \mathsf{trg}(\delta) = \mathsf{trg}(\delta') \quad \Rightarrow \quad \delta = \delta',$$

and *deterministic* if it satisfies the stronger property

$$\mathsf{src}(\delta) = \mathsf{src}(\delta') \wedge \mathsf{act}(\delta) = \mathsf{act}(\delta') \quad \Rightarrow \quad \delta = \delta'.$$

We say that $\mathcal{A}$ is a *finite automaton* if both Q and E are finite. An *execution fragment* of a lean[1] LTS $\mathcal{A}$ is a finite or infinite alternating sequence $q_0 a_1 q_1 a_2 q_2 \cdots$ of states and actions of $\mathcal{A}$, beginning with a state, and if it is finite also ending with a state, such that for all $i > 0$, $q_{i-1} \xrightarrow{a_i} q_i$. An *execution* of $\mathcal{A}$ is an execution fragment that begins with the initial state of $\mathcal{A}$. A state q of $\mathcal{A}$ is *reachable* if it is the last state of some finite execution of $\mathcal{A}$. For $\gamma = q_0 a_1 q_1 a_2 q_2 \cdots$ an execution fragment of $\mathcal{A}$, $\mathit{trace}(\gamma)$ is defined as the sequence $a_1 a_2 \cdots$. We write $q \xrightarrow{\sigma} q'$ if $\mathcal{A}$ has a finite execution fragment γ with $\mathit{first}(\gamma) = q$, $\mathit{last}(\gamma) = q'$, and $\mathit{trace}(\gamma) = \sigma$. We say that σ is a *trace* of q, and write $q \xrightarrow{\sigma}$, if there exists a q' such that $q \xrightarrow{\sigma} q'$. Moreover, σ is a *trace* of $\mathcal{A}$ if it is a trace of the initial state of $\mathcal{A}$. We write $\mathit{traces}^*(q)$ for the set of traces of q. We say that σ is a *distinguishing trace* of q and q' if either it is a trace of q but not of q', or the other way around.

[1]For non lean LTS's this notion of execution fragments is less natural since it does not record all information about the dynamic behavior of such systems.

The main equivalence relation between LTS's that we consider in this paper is bisimulation equivalence: according to our definition in Section 5 an Implementation Under Test (IUT) conforms to a specification *Spec* iff certain associated LTS's are bisimilar. However, as a consequence of the fact that these LTS's are deterministic, the reader may equally well think of conformance in terms of trace equivalence, and view bisimulations as a convenient characterization of trace equivalence.

Definition 2.2. Let $\mathcal{A}$ be an LTS. A relation $R \subseteq Q_{\mathcal{A}} \times Q_{\mathcal{A}}$ is a *bisimulation on* $\mathcal{A}$ iff

- $R(q_1, q_2)$ and $q_1 \xrightarrow{a} q_1'$ implies that there is a $q_2' \in Q_{\mathcal{A}}$ such that $q_2 \xrightarrow{a} q_2'$ and $R(q_1', q_2')$,
- $R(q_1, q_2)$ and $q_2 \xrightarrow{a} q_2'$ implies that there is a $q_1' \in Q_{\mathcal{A}}$ such that $q_1 \xrightarrow{a} q_1'$ and $R(q_1', q_2')$.

States q, q' of $\mathcal{A}$ are *bisimilar*, notation $q \simeq_{\mathcal{A}} q'$, if there exists a bisimulation R on $\mathcal{A}$ with $R(q, q')$.

States q, q' of LTS's $\mathcal{A}$ and $\mathcal{A}'$, respectively, are *bisimilar*, notation $\mathcal{A}, \mathcal{A}' : q \simeq q'$, if there exists a bisimulation R on the disjoint union of $\mathcal{A}$ and $\mathcal{A}'$ (with arbitrary initial state) that relates q to q'. LTS's $\mathcal{A}$ and $\mathcal{A}'$ are *bisimilar*, notation $\mathcal{A} \simeq \mathcal{A}'$, if $\mathcal{A}, \mathcal{A}' : q^0_{\mathcal{A}} \simeq q^0_{\mathcal{A}'}$. □

It is well known that if $\mathcal{A}$ is deterministic, for all states q, q' of $\mathcal{A}$, $\mathcal{A} : q \simeq q'$ iff $traces^*(q) = traces^*(q')$. As a consequence, two deterministic LTS's $\mathcal{A}$ and $\mathcal{A}'$ are bisimilar iff they have the same sets of traces.

2.3 Bounded Time Domain Automata

In this subsection we recall the bounded time domain automata model from [SV96], which is a variant of the timed automata model of Alur and Dill [AD94]. In the Alur-Dill model clocks range over $\mathsf{R}^{\geq 0}$, and the only assignments that are allowed are *clock resets* of the form $x := 0$. This in contrast to the BTDA model, where the domain $dom(x)$ of a clock x is the union of a bounded interval and the singleton $\{\infty\}$. Intuitively, the value of x is only relevant when contained in the interval: beyond the upper bound of the interval one only knows that the value of x is "large". The BTDA model also allows for more general assignments of the form $x := n$ or $x := y + n$, for x and y clocks and $n \in \mathsf{Z}^\infty$.

As shown in [SV96], the BTDA model is essentially equivalent to the Alur-Dill model but often allows for more compact representations of timed systems. Also, it turns out that the use of ∞ simplifies the technical development in the rest of this paper.

Below we first define the auxiliary concepts of clocks and constraints, before proceeding to the definition of BTDA's and their operational semantics.

2.3.1 Clocks and Constraints

A *clock* is a variable x with a domain $dom(x)$ of the form $J \cup \{\infty\}$, where J is an interval over R with infimum and supremum in Z. Let C be a finite set of clocks. We write $intv(x) \triangleq dom(x) - \{\infty\}$. A *term over* C is an expression generated by the grammar $e ::= x \mid n \mid e + n$, where x is a clock in C and $n \in \mathsf{Z}^\infty$. We denote the set of all such terms by $T(C)$. A *constraint over* C is a Boolean combination φ of inequalities of the form $e \leq e'$ or $e < e'$ with $e, e' \in T(C)$. We denote the set of all such formulas by $F(C)$. The Boolean constants T and F, denoting truth and falsehood, respectively, as well as equations of the form $x = n$ are definable by constraints. In fact, for each term e and each interval J with integer bounds, the predicate $e \in J$ can be expressed as a constraint. A *(simultaneous) assignment over* C is a function μ from C to $T(C)$. We denote the set of all such assignments by $M(C)$, and (for instance) write $x := 5$ for the assignment μ with $\mu(x) = 5$.

A *clock valuation over* C is a map v that assigns to each clock $x \in C$ a value in its domain. With $V(C)$ we denote the set of clock valuations over C. In the obvious way, a clock valuation v is lifted to a function $\overline{v}$ that takes a term and returns a value. We say that v *satisfies* φ, notation $v \models \varphi$, if φ evaluates to true under valuation v. A constraint φ is *satisfiable* if there is a valuation

v such that $v \models \varphi$; constraint φ *holds* if for all valuations v, $v \models \varphi$. If $d \in \mathsf{R}^{\geq 0}$ then $v \oplus d$ is the clock valuation defined by

$$(v \oplus d)(x) \quad \triangleq \quad \begin{cases} v(x) + d & \text{if } v(x) + d \in intv(x) \\ \infty & \text{otherwise} \end{cases}$$

The *hull* of φ is the set of clock valuations v that satisfy, for all $d \in \mathsf{R}^{\geq 0}$, $v \oplus d \models \varphi$ iff $d = 0$. The *interior* of φ is the set of all valuations that satisfy φ but are not in its hull. So if a clock valuation v is in the hull of φ, then any non-zero increment of the value of clocks under v will violate φ. For each constraint φ, let $\mathit{hull}(\varphi)$ be a constraint such that, for all v, $v \models \mathit{hull}(\varphi)$ iff v is in the hull of φ. Similarly, let $\mathit{interior}(\varphi)$ be a constraint such that, for all v, $v \models \mathit{interior}(\varphi)$ iff v is in the interior of φ. It is not hard to see that such constraints always exist and can be effectively computed. For example, $\mathit{hull}(x \leq 5) = (x = 5)$, $\mathit{hull}(x < 5) = \mathsf{F}$, and $\mathit{interior}(x \leq 5) = (x < 5) = \mathit{interior}(x < 5)$.

2.3.2 BTDA's and Their Operational Semantics

A bounded time domain automaton is a finite automaton together with some annotations to restrict real-time behavior. To start with, a set of clocks is associated with the automaton. Each clock gets an initial value, and when time advances with an amount d, the value of all clocks is incremented uniformly (according to $\oplus$) with d. To each state we associate an invariant; we require that the automaton may only reside in a state as long as the invariant remains true. In addition, a clock constraint is associated to each transition; we require that a transition may be taken only if the current valuation of the clocks satisfies this constraint. When a transition occurs, the clock values are updated according to a given assignment. We require that in the new state each clock again takes a value in its domain. All this is formalized in the two following definitions.

Definition 2.3. A *timing annotation* for an automaton $\mathcal{A}$ is a tuple $\mathcal{T} = (C, \mathit{Inv}, G, A, v^0)$, where

- C is a finite set of clocks.
- $\mathit{Inv} : Q \to F(C)$ associates an *invariant* to each state.
- $G : E \to F(C)$ associates a *guard* to each transition.
- $A : E \to M(C)$ associates an assignment to each transition. We require that, for each $\delta \in E$, the constraint $\mathit{Inv}(\mathsf{src}(\delta)) \wedge G(\delta) \Rightarrow \bigwedge_{x \in C}(A(\delta)(x) \in \mathit{dom}(x))$ holds.
- $v^0 \in V(C)$ is the *initial valuation.* We require that $v^0 \models \mathit{Inv}(q^0)$ and, for all x, $v^0(x) \in \mathsf{Z}^\infty$.

A *bounded time domain automaton (BTDA)* is a pair $\mathcal{B} = (\mathcal{A}, \mathcal{T})$, where $\mathcal{A}$ is a finite automaton with $\Sigma_{\mathcal{A}} \cap \mathsf{R}^{>0} = \emptyset$, and $\mathcal{T}$ is a timing annotation for $\mathcal{A}$. We write $Q_{\mathcal{B}}$, $E_{\mathcal{B}}$, $C_{\mathcal{B}}$, etc., for the components of $\mathcal{A}$ and $\mathcal{T}$. □

Definition 2.4. The *operational semantics* $\mathcal{OS}(\mathcal{B})$ of a BTDA $\mathcal{B}$ is the lean LTS $\mathcal{A}$ which, up to identity of transitions, is specified by

$$\begin{aligned} S_{\mathcal{A}} &= \{(q, v) \in Q_{\mathcal{B}} \times V(C_{\mathcal{B}}) \mid v \models \mathit{Inv}_{\mathcal{B}}(q)\} \\ \Sigma_{\mathcal{A}} &= \Sigma_{\mathcal{B}} \cup \mathsf{R}^{>0} \\ s^0_{\mathcal{A}} &= (q^0_{\mathcal{B}}, v^0_{\mathcal{B}}) \end{aligned}$$

and $\to_{\mathcal{A}}$ is the smallest predicate that satisfies the following two rules, for all $(q, v), (q', v') \in S_{\mathcal{A}}$, $a \in \Sigma_{\mathcal{B}}$, $\delta \in E_{\mathcal{B}}$ and $d \in \mathsf{R}^{>0}$,

$$\frac{\delta : q \xrightarrow{a} q', \quad v \models G_{\mathcal{B}}(\delta), \quad v' = \overline{v} \circ A_{\mathcal{B}}(\delta)}{(q, v) \xrightarrow{a}_{\mathcal{A}} (q', v')}$$

$$\frac{q = q', \quad v' = v \oplus d, \quad \forall 0 \leq d' \leq d : v \oplus d' \models Inv_{\mathcal{B}}(q)}{(q, v) \xrightarrow{d}_{\mathcal{A}} (q', v')}$$

The actions in $\mathsf{R}^{>0}$ are referred to as *time delays*. In order not to confuse the states of a BTDA with those of its operational semantics, we will refer to the states of a BTDA $\mathcal{B}$ as *locations*. We will use $q, ..$ to range over locations, and $r, s, ..$ to range over the states of the operational semantics $\mathcal{OS}(\mathcal{B})$. We write $S_{\mathcal{B}}$ for the set of states of $\mathcal{OS}(\mathcal{B})$, and $s^0_{\mathcal{B}}$ for the initial state of $\mathcal{OS}(\mathcal{B})$. □

2.4 Timed I/O Automata

In this subsection, we define the model of timed I/O automata (TIOA's) as an extension of the BTDA model in which the actions are partitioned into input and output actions. We impose some restrictions in order to ensure "testability" of the model. Intuitively (a formal definition will be presented in Section 3), an *experiment* or *test sequence* for a TIOA is a finite sequence of delays and input actions that can be applied to the TIOA. In order to fully test a TIOA by test sequences the TIOA should be *controllable* in the sense that it should be possible for an environment to drive a TIOA through all of its transitions. An obvious prerequisite for controllability is that a TIOA is *deterministic*. However this is not enough. We also need to require that a TIOA has *isolated outputs*: for each state, if an output is enabled then no other input or output transition is enabled. In this way we exclude that a TIOA can autonomously choose between performing different outputs, or between performing an output and accepting an input.

Since input actions are under control of the environment of a TIOA, a TIOA should always accept inputs. Traditionally [LT87, SGSL98], this leads to the requirement that every input is enabled in every state. This however is in conflict with the condition of isolated outputs. We therefore impose a slightly weaker *input enabling* condition: each input is enabled only in the interior of the invariant of each location. This means that inputs are enabled as long as time can progress. Since inputs and outputs are mutually exclusive, this ensures that a TIOA cannot choose to pass over output actions by letting time pass. Together, the conditions of determinism, isolated outputs and input enabling ensure that a TIOA is controllable.

According to our definitions, there are BTDA's in which from some (or even all) states no outgoing execution fragment exists in which the sum of the time delays diverges. It may for instance occur that a state has no outgoing transition at all ("time deadlock"), or that there is an infinite sequence of consecutive output actions without any time delays in between ("Zeno behavior"). Since (we believe) these behaviors can not occur in the real world and we need to exclude them in order to develop our testing approach, we demand as a final requirement that a TIOA is *progressive*: from each state there should be an outgoing execution fragment containing no input actions in which the sum of the time delays diverges. Progressiveness implies that in each state on the hull of an invariant an output action is enabled. Moreover, after a finite number of consecutive output actions time will be allowed to advance and, consequently, input actions will be enabled again. As a result, a TIOA can never preempt input actions indefinitely by performing output actions. So although within our model input actions are not enabled in every *state*, they are accepted at every *time instance*.

Definition 2.5. A *timed I/O automaton (TIOA)* is a pair $\mathcal{M} = (\mathcal{B}, \mathcal{P})$, where $\mathcal{B}$ is a BTDA and $\mathcal{P} = (I, O)$ is a partitioning of $\Sigma_{\mathcal{B}}$ in *input actions* and *output actions*. We require that the following properties hold, for all $\delta, \delta' \in E$, $q \in Q$, and $i \in I$:

1. (Determinism) if $\mathsf{src}(\delta) = \mathsf{src}(\delta')$, $\mathsf{act}(\delta) = \mathsf{act}(\delta')$ and $G(\delta) \wedge G(\delta')$ is satisfiable then $\delta = \delta'$
2. (Isolated outputs) if $\mathsf{src}(\delta) = \mathsf{src}(\delta')$, $\mathsf{act}(\delta) \in O$ and $G(\delta) \wedge G(\delta')$ is satisfiable then $\delta = \delta'$
3. (Input enabling) $interior(Inv(q)) \Rightarrow \bigvee_{\delta \in \mathsf{from}(q,i)} G(\delta)$ holds, where $\mathsf{from}(q, i) \triangleq \{\delta \in E \mid \mathsf{src}(\delta) = q \wedge \mathsf{act}(\delta) = i\}$
4. (Progressiveness) For every state of $\mathcal{OS}(\mathcal{B})$ there exists an infinite execution fragment that starts in this state, contains no input actions, and in which the sum of the delays diverges

We write $Q_{\mathcal{M}}$, $\Sigma_{\mathcal{M}}$, $C_{\mathcal{M}}$, $Inv_{\mathcal{M}}$, $I_{\mathcal{M}}$, etc., for the components of $\mathcal{B}$ and $\mathcal{P}$. The *operational semantics* $\mathcal{OS}(\mathcal{M})$ of $\mathcal{M}$ is just the operational semantics of the contained BTDA $\mathcal{B}$. Moreover, we write $\mathcal{M} \simeq \mathcal{M}'$ whenever $\mathcal{OS}(\mathcal{M}) \simeq \mathcal{OS}(\mathcal{M}')$. □

The following lemma, which is a direct corollary of the definitions, gives four basic properties of the operational semantics of a timed I/O automaton.

Lemma 2.6. *Let $\mathcal{M}$ be a TIOA. Then*

1. *$\mathcal{OS}(\mathcal{M})$ is deterministic.*
2. *$\mathcal{OS}(\mathcal{M})$ has Wang's [Yi90] time additivity property: $s \xrightarrow{d+d'} s' \Leftrightarrow \exists r : s \xrightarrow{d} r \wedge r \xrightarrow{d'} s'$.*
3. *Each state of $\mathcal{OS}(\mathcal{M})$ has either (a) a single outgoing transition labeled with an output action, or (b) both outgoing delay transitions and outgoing input transitions (one for each input action), but no outgoing output transitions. States of type (b) are called* stable.
4. *For each state $s \in S_{\mathcal{M}}$, there exists a unique finite sequence of output actions σ and a unique stable state s' such that $s \xrightarrow{\sigma} s'$.*

Example 2.7. Consider the automaton represented in Figure 1. It denotes a switch that can be turned on at any time and switches off automatically 5 time units after the last time it has been turned on.

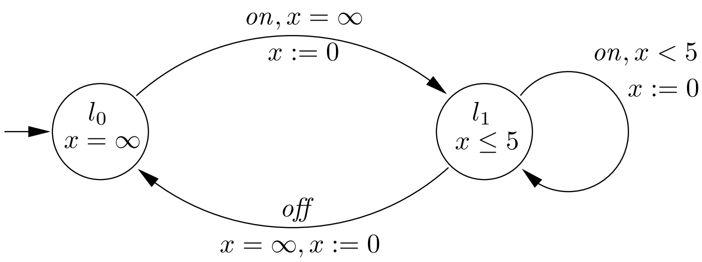

Figure 1: A switch

The switch can be described formally as a TIOA $\mathcal{M} = (\mathcal{B}, \mathcal{P})$, where $\mathcal{B} = (\mathcal{A}, \mathcal{T})$ is a BTDA, in the following way. First, the finite automaton $\mathcal{A} = (Q, E, \Sigma, \mathsf{src}, \mathsf{act}, \mathsf{trg}, q^0)$ is given by

- $Q = \{l_0, l_1\}$
- $E = \{\delta_0, \delta_1, \delta_2\}$
- $\Sigma = \{on, off\}$
- $\mathsf{src}(\delta_0) = l_0$, $\mathsf{src}(\delta_1) = \mathsf{src}(\delta_2) = l_1$
- $\mathsf{act}(\delta_0) = \mathsf{act}(\delta_2) = on$, $\mathsf{act}(\delta_1) = off$
- $\mathsf{trg}(\delta_0) = \mathsf{trg}(\delta_2) = l_1$, $\mathsf{trg}(\delta_1) = l_0$
- $q^0 = l_0$

Secondly, the timing annotation $\mathcal{T} = (C, Inv, G, A, v^0)$ is given by

- $C = \{x\}$ with $dom(x) = [0, 5] \cup \{\infty\}$

- $Inv(l_0) = (x = \infty)$, $Inv(l_1) = (x \leq 5)$
- $G(\delta_0) = (x = \infty)$, $G(\delta_1) = (x = 5)$, $G(\delta_2) = (x < 5)$
- $A(\delta_0) = (x := 0)$, $A(\delta_1) = (x := \infty)$, $A(\delta_2) = (x := 0)$
- $v^0(x) = \infty$

Thirdly, the partitioning $\mathcal{P} = (I, O)$ consists of $I = \{on\}$ and $O = \{off\}$. It is not difficult to check that $\mathcal{M}$ is indeed a TIOA.

In location l_0 clock x is not used; therefore x has been given the value ∞. The value of x becomes relevant as soon as the input action *on* occurs and the transition to location l_1 is made. After this transition, the input action *on* is enabled in the interior of $Inv(l_1)$. Recall that $hull(x \leq 5) = (x = 5)$ and $interior(x \leq 5) = (x < 5)$. As soon as 5 time units have passed after the last *on* action, the hull of the invariant is reached, time cannot advance any longer, and the switch automaton performs its output action *off* to return to its initial state.

Example 2.8. Timed I/O automata form a natural generalization of the classical Mealy machines [Koh78]. Recall that a Mealy machine is a tuple $\mathcal{F} = (I, O, Q, \delta, \lambda, q^0)$, where I, O, Q are finite, nonempty sets of inputs, outputs, and states, respectively,

- $\delta : I \times Q \to Q$ is the transition function,
- $\lambda : I \times Q \to O$ is the output function, and
- q^0 is the initial state.

To a Mealy machine $\mathcal{F}$ we associate a timed I/O automaton $\mathcal{M}$ with locations $Q \cup (I \times Q)$, inputs I, outputs O, and initial location q^0. For each state $q \in Q$ and input action $i \in I$, we introduce a pair of transitions

$$q \xrightarrow{i} (i, q) \quad \text{and} \quad (i, q) \xrightarrow{\lambda(i,q)} \delta(i, q).$$

We equip $\mathcal{M}$ with a single clock x with domain $\{0, \infty\}$. In order to model that Mealy machines accept inputs at any time, a constraint $x = \infty$ is associated to each location $q \in Q$ and to each input transition $q \xrightarrow{i} (i, q)$. To capture the intuition that in a Mealy machine each input is immediately followed by an output, a constraint $x = 0$ is associated to each location $(i, q) \in I \times Q$ and to each output transition $(i, q) \xrightarrow{o} q'$. Finally, to make $\mathcal{M}$ into a proper TIOA, we assign ∞ as initial value to x, annotate each input transition with an assignment $x := 0$, and each output transition with an assignment $x := \infty$.

Besides the above translation from Mealy machines to TIOA's, many other possible translations exist. The TIOA model allows one to express, for instance, that some amount of time may elapse in between an input and the subsequent output. In this case one has to specify what happens if another input arrives before the output is produced. One possibility here is to jump to a newly added error state, but one may also decide to ignore such an input.

3 Experiments

We view timed I/O automata as machines on which one can do experiments. An *experiment* or *test sequence* for a TIOA $\mathcal{M}$ is a finite sequence of delays and input actions of $\mathcal{M}$. We denote the set of all experiments for $\mathcal{M}$ by $Exp_{\mathcal{M}}$. An experiment σ can be applied to the machine $\mathcal{M}$ starting from any state s. The application of σ to $\mathcal{M}$ in s uniquely determines a finite, maximal execution fragment in $\mathcal{OS}(\mathcal{M})$. The existence of such an execution fragment is guaranteed by the properties that we demand of TIOA's. For black box testing, one is only interested in the observable behavior induced by the execution, that is, actions and passage of time, but not states. For instance, if the experiment *on* 6 —which may be read as "press input *on* and observe the system during 6 units of time"— is applied to the initial state of Example 2.7, it determines a unique execution whose observable behavior is the trace *on* 5 *off* 1, that is, "after pressing input *on* and waiting for 5 time units, the switch turns automatically *off*; in the next unit of time nothing happens".

In the following we formally define what it means to perform an experiment, and we prove that an experiment induces a unique execution fragment. The outcome of performing an experiment on $\mathcal{M}$ is described in terms of an auxiliary labeled transition system $\mathcal{E}_{\mathcal{M}}$.

Definition 3.1. The *experiment LTS* $\mathcal{E}_{\mathcal{M}}$ is the lean LTS with $Exp_{\mathcal{M}} \times S_{\mathcal{M}}$ as its set of states, $\Sigma_{\mathcal{M}}$ as its set of actions, $(\epsilon, s^0_{\mathcal{M}})$ as its (arbitrarily chosen) initial state, and a transition relation $\to$ that is inductively defined as the least relation satisfying the following four rules, for all $s, s' \in S_{\mathcal{M}}$, $\sigma \in Exp_{\mathcal{M}}$, $i \in I_{\mathcal{M}}$, $o \in O_{\mathcal{M}}$ and $d, d' \in \mathsf{R}^{>0}$,

$$\frac{s \xrightarrow{o}_{\mathcal{OS}(\mathcal{M})} s'}{(\sigma, s) \xrightarrow{o} (\sigma, s')} \tag{1}$$

$$\frac{s \xrightarrow{i}_{\mathcal{OS}(\mathcal{M})} s'}{(i\,\sigma, s) \xrightarrow{i} (\sigma, s')} \tag{2}$$

$$\frac{s \xrightarrow{d}_{\mathcal{OS}(\mathcal{M})} s'}{(d\,\sigma, s) \xrightarrow{d} (\sigma, s')} \tag{3}$$

$$\frac{s \xrightarrow{d'}_{\mathcal{OS}(\mathcal{M})} s', \quad \sup\{t \in \mathsf{R}^{>0} \mid s \xrightarrow{t}_{\mathcal{OS}(\mathcal{M})}\} = d' < d}{(d\,\sigma, s) \xrightarrow{d'} (d - d'\,\sigma, s')} \tag{4}$$

□

The following theorem basically says that for each experiment and each state there is a unique corresponding finite execution fragment in the experiment automaton.

Theorem 3.2. *Let $\mathcal{M}$ be a TIOA. Then*

1. *each state of $\mathcal{E}_{\mathcal{M}}$ has at most one outgoing transition, and*
2. *$\mathcal{E}_{\mathcal{M}}$ does not have an infinite execution fragment.*

Proof. Part (1) follows directly from the definition of $\mathcal{E}_{\mathcal{M}}$ together with Lemma 2.6.

Part (2) is proved by contradiction. Suppose that β is an infinite execution fragment of $\mathcal{E}_{\mathcal{M}}$. Because experiments have a finite length, transitions of type (2) and type (3) reduce the length of the experiment, and the two other types of transitions leave the length of experiments unchanged, β has an (infinite) suffix β' that contains no transitions of type (2) and type (3). If we project all states in β' on their second component, then we obtain an infinite execution fragment γ of the LTS $\mathcal{OS}(\mathcal{M})$ that contains no input actions, and in which the sum of the delays converges. Let s be the first state of γ. Since $\mathcal{M}$ is progressive, $\mathcal{OS}(\mathcal{M})$ has an infinite execution fragment γ' that starts in s, that contains no input actions, and in which the sum of the delays diverges. Let $\gamma = s_0 a_1 s_1 a_2 s_2 \cdots$ and let $\gamma' = s'_0 a'_1 s'_1 a'_2 s'_2 \cdots$. Then $s = s_0 = s'_0$. Inductively, we construct a monotonic function $f : \mathsf{N} \to \mathsf{N}$ that satisfies, for all $i \in \mathsf{N}$, the following two properties:

1. $s_i = s'_{f(i)}$
2. $s'_{f(i)} \xrightarrow{a_{i+1}} s'_{f(i+1)}$

For the induction base, we define $f(0) = 0$. Since both γ and γ' start with s, we have $s_0 = s = s'_{f(0)}$. Now suppose that f has been defined for all $j \leq k$ and $s_k = s'_{f(k)}$. In order to define $f(k+1)$ we distinguish between two cases.

1. a_{k+1} is an output action. In this case define $f(k+1) = f(k) + 1$. Using Lemma 2.6(3) we obtain $s_{k+1} = s'_{f(k+1)}$ and $s'_{f(k)} \stackrel{a_{k+1}}{\rightarrow} s'_{f(k+1)}$.

2. a_{k+1} is a delay. Then the transition $s_k \stackrel{a_{k+1}}{\rightarrow} s_{k+1}$ originates from a transition of type (4) in $\mathcal{E}_{\mathcal{M}}$. This implies that a_{k+1} is the maximal delay transition that is enabled in s_k. Using the fact that γ' diverges and Lemma 2.6, we can infer that there exists an index $m > f(k)$ such that all actions a'_j for $f(k) < j \leq m$ are delays, with a sum equal to a_{k+1}, and $s_{k+1} = s'_m$. Now define $f(k+1) = m$. Clearly $s_{k+1} = s'_{f(k+1)}$ and $s'_{f(k)} \stackrel{a_{k+1}}{\rightarrow} s'_{f(k+1)}$.

From the construction together with Lemma 2.6 it follows that, for all i, the sum $D(i)$ of the delays in $s_0 a_1 s_1 a_2 s_2 \cdots s_i$ is equal to the sum $D'(i)$ of the delays in the fragment $s'_0 a'_1 s'_1 a'_2 s'_2 \cdots s'_{f(i)}$. Since f is monotonic and the sum of the delays in γ' diverges, the value of $D'(i)$ increases without bound if $i \rightarrow \infty$. This contradicts the fact that the sum of the delays in the experiment part of s gives a finite upper bound for all the sums $D(i)$. ⊠

Theorem 3.2(1) allows us to define $exec_{\mathcal{M}}(\sigma, s)$ as the execution fragment of $\mathcal{OS}(\mathcal{M})$ obtained by projecting the states in the unique maximal execution fragment of $\mathcal{E}_{\mathcal{M}}$ that starts in (σ, s) on their second component. Theorem 3.2(2) implies that it is a finite execution fragment. Write $s \stackrel{\sigma}{\Rightarrow} s'$ if s' is the last state of $exec_{\mathcal{M}}(\sigma, s)$ (note that s' is stable). For instance, the application of the experiment $\sigma = on\,6$ to the switch of Example 2.7 from its initial state determines the following execution

$$exec_{\mathcal{M}}(\sigma, (l_0, x{=}\infty)) \;=\; (l_0, x{=}\infty)\; on\; (l_1, x{=}0)\; 5\; (l_1, x{=}5)\; off\; (l_0, x{=}\infty)\; 1\; (l_0, x{=}\infty).$$

Hence, $(l_0, x = \infty) \stackrel{\sigma}{\Rightarrow} (l_0, x = \infty)$. As we said, for black box testing, one is only interested in observable behavior, not in states. We define $outcome_{\mathcal{M}}(\sigma, s)$, the *outcome of experiment* σ *in state* s *of* $\mathcal{M}$, as the trace of the execution fragment that is induced by performing the experiment:

$$outcome_{\mathcal{M}}(\sigma, s) \quad \triangleq \quad trace(exec_{\mathcal{M}}(\sigma, s))$$

In this way, the outcome of the previous experiment is $outcome_{\mathcal{M}}(\sigma, (l_0, x{=}\infty)) \;=\; on\,5\; off\,1$.

We end this section with a small lemma stating that each trace σ that leads from a given state s to a stable state s' can be retrieved as the outcome of the experiment obtained by projecting σ on input actions and delays.

Lemma 3.3. *Suppose* $s \stackrel{\sigma}{\rightarrow} s'$, s' *is stable, and* $\sigma' = \sigma\lceil(I_{\mathcal{M}} \cup \mathsf{R}^{>0})$. *Then* $outcome_{\mathcal{M}}(\sigma', s) = \sigma$ *and* $s \stackrel{\sigma'}{\Rightarrow} s'$.

Proof. Let γ be the unique execution fragment of $\mathcal{OS}(\mathcal{M})$ with $first(\gamma) = s$, $last(\gamma) = s'$, and $trace(\gamma) = \sigma$. By induction on the number n of transitions in γ we prove $exec_{\mathcal{M}}(\sigma', s) = \gamma$.

Suppose $n = 0$. Then $s = s' = \gamma$ and $\sigma = \sigma' = \epsilon$. Since s is stable, state (ϵ, s) of $\mathcal{E}_{\mathcal{M}}$ has no outgoing transitions. Thus $exec_{\mathcal{M}}(\sigma', s) = s = \gamma$.

For the induction step, suppose that $n > 0$. Let $s \stackrel{a}{\rightarrow} s''$ be the first transition of γ, and let γ' be the unique execution fragment satisfying $\gamma = s\; a\; \gamma'$. We distinguish between two cases:

1. a is an output action. Then, by rule (1), $\mathcal{E}_{\mathcal{M}}$ contains a transition $(\sigma', s) \stackrel{a}{\rightarrow} (\sigma', s'')$. By induction hypothesis $exec_{\mathcal{M}}(\sigma', s'') = \gamma'$. Since clearly $exec_{\mathcal{M}}(\sigma', s) = s\; a\; exec_{\mathcal{M}}(\sigma', s'')$, we infer $exec_{\mathcal{M}}(\sigma', s) = \gamma$.

2. a is an input or delay action. Then σ' is of the form $a\; \sigma''$. Hence, by application of rule (2) or rule (3), respectively, $\mathcal{E}_{\mathcal{M}}$ contains a transition $(\sigma', s) \stackrel{a}{\rightarrow} (\sigma'', s'')$. By induction hypothesis $exec_{\mathcal{M}}(\sigma'', s'') = \gamma'$. Since clearly $exec_{\mathcal{M}}(\sigma', s) = s\; a\; exec_{\mathcal{M}}(\sigma'', s'')$, we infer $exec_{\mathcal{M}}(\sigma', s) = \gamma$.

Now the result follows since $outcome_{\mathcal{M}}(\sigma', s) = trace(exec_{\mathcal{M}}(\sigma', s)) = trace(\gamma) = \sigma$. ⊠

4 Discretization of the State Space

Even though our experiments are very simple, the set $Exp_{\mathcal{M}}$ of experiments for a given TIOA $\mathcal{M}$ is uncountably large, due to the possible occurrence of real numbers within experiments. Also the LTS $\mathcal{OS}(\mathcal{M})$, which gives the operational behavior of $\mathcal{M}$, is a highly infinite object. It is thus unclear how we should select a finite collection of tests if we want to establish that an IUT conforms to a specification $\mathcal{M}$. Fortunately however, the technical results of this section will enable us to restrict attention to a finite subautomaton of $\mathcal{OS}(\mathcal{M})$ which contains enough information to characterize $\mathcal{OS}(\mathcal{M})$ itself. We call it a *grid automaton*. In fact, for each pair $\mathcal{M}$, $\mathcal{M}'$ of TIOA's, we can effectively find two grid automata such that $\mathcal{M}$ is bisimilar to $\mathcal{M}'$ iff the grid automata are bisimilar. In this way, whenever we want to establish that some black box conforms to the original TIOA specification, we can restrict attention to their grid automata. Using the fact that, in the context of deterministic machines, bisimulation coincides with trace equivalence, checking bisimulation reduces to the application of an appropriate set of experiments. The grid automata can be fully and effectively explored by a finite number of experiments in $Exp_{\mathcal{M}}$, using standard techniques for testing finite automata.

4.1 Regions

Our construction of a finite subautomaton uses the fundamental concept of a *region*, due to Alur & Dill [AD94]. The key idea behind the definition of a region is that, even though the number of states of an LTS $\mathcal{OS}(\mathcal{M})$ is infinite, not all of these states are distinguishable via constraints. If two states corresponding to the same location agree on the integral parts of all the clock values, and also on the ordering of the fractional parts of all the clocks, then these two states cannot be distinguished by constraints.

Definition 4.1. The equivalence relation $\cong$ over the set $V(C)$ of valuations of a set C of clocks is given by: $v \cong v'$ iff, for all $x, y \in C$,

1. $v(x) = \infty$ iff $v'(x) = \infty$,
2. if $v(x) \neq \infty$ then $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$ and ($\mathit{fract}(v(x)) = 0$ iff $\mathit{fract}(v'(x)) = 0$),
3. if $v(x) \neq \infty \neq v(y)$ then $\mathit{fract}(v(x)) \leq \mathit{fract}(v(y))$ iff $\mathit{fract}(v'(x)) \leq \mathit{fract}(v'(y))$.

A *region* is an equivalence class of valuations induced by $\cong$. □

Lemma 4.2. *If $v \cong v'$ then $v \models \varphi \Leftrightarrow v' \models \varphi$.*

The equivalence relation $\cong$ on the clock valuations of a TIOA $\mathcal{M}$ is lifted to an equivalence relation $\cong$ on $S_{\mathcal{M}}$ by defining

$$(q, v) \cong (q', v') \quad \triangleq \quad q = q' \wedge v \cong v'$$

A *region* of $\mathcal{M}$ is an equivalence class of states induced by $\cong$. Similarly, for $\mathcal{M}_1$ and $\mathcal{M}_2$ TIOA's with clocks C_1 and C_2, respectively, the equivalence relation $\cong$ on $V(C_1 \cup C_2)$ (w.l.o.g. we assume that C_1 and C_2 are disjoint) is lifted to an equivalence relation $\cong$ on $S_{\mathcal{M}_1} \times S_{\mathcal{M}_2}$ by defining

$$((q_1, v_1), (q_2, v_2)) \cong ((q_1', v_1'), (q_2', v_2')) \quad \triangleq \quad q_1 = q_1' \wedge q_2 = q_2' \wedge v_1 \cup v_2 \cong v_1' \cup v_2'$$

A *region* of $\mathcal{M}_1$ and $\mathcal{M}_2$ is an equivalence class of pairs of states induced by $\cong$. Note that $(r_1, r_2) \cong (s_1, s_2)$ implies $r_1 \cong s_1 \wedge r_2 \cong s_2$, but that the converse implication does not hold in general.

Alur & Dill [AD94] show that for a set of clocks C the number of regions of $V(C)$ is bounded by $\mid C \mid ! \cdot 2^{|C|} \cdot \Pi_{x \in C}(2c_x + 2)$, where for each clock x, c_x denotes the length of the domain interval $\mathit{intv}(x)$. This means that also the number of regions of a TIOA is (in the worst case) exponential in the number of clocks. In practice the use of invariants may keep the number of regions small. The switch TIOA of Example 2.7 has 12 regions, and the TIOA associated to a Mealy machine in Example 2.8 has $\mid Q \mid \cdot (\mid I \mid + 1)$ regions.

4.2 Uniform mappings

The concept of a *uniform mapping* was introduced by Čerāns [Čer92b, Čer92a]. Uniform mappings provide a convenient characterization of regions. They play a central role in Čerāns' proof that bisimulation equivalence is decidable for timed automata, and are also used heavily in this section.

Definition 4.3. A continuous mapping[2] $\rho : \mathsf{R}^\infty \to \mathsf{R}^\infty$ is *uniform* if

1. ρ is strongly monotone (so $t > u$ implies $\rho(t) > \rho(u)$),
2. $\rho(0) = 0$,
3. $\rho(t + n) = \rho(t) + n$, for every real number t and integer n.

A uniform mapping ρ is extended in a homomorphic manner to any structure containing elements of R^∞. In particular, for any clock valuation v, $\rho(v)$ is equal to the function v' given by $v'(x) \triangleq \rho(v(x))$, for all x. □

Note that conditions 1, 2 and 3 in Definition 4.3 together imply that $\rho(n) = n$, for all $n \in \mathsf{Z}^\infty$. Below we rephrase the basic results of Čerāns [Čer92b, Čer92a] about uniform mappings in our setting. We first need to prove five technical lemmas to prepare for the main results of this subsection, which say that uniform mappings "preserve" the transition relation. The proofs of Lemmas 4.4, 4.6 and 4.7 easily follow from the definitions. The proofs of Lemmas 4.5 and 4.8 are somewhat more tricky and therefore outlines have been included below.

Lemma 4.4. *Suppose v is a clock valuation over a set of clocks C and ρ is a uniform mapping. Then $\rho(v)$ is a clock valuation over C.*

Lemma 4.5. *$v \cong v'$ iff there exists a uniform mapping ρ, such that $v' = \rho(v)$.*

Proof. "$\Leftarrow$" Routine checking. Use the observation that, for each uniform mapping ρ, the inverse mapping ρ^{-1} is defined and also uniform.

"$\Rightarrow$" Let $C = \{x_1, \ldots, x_n\}$. We order the clocks according to the value of their fractional part in v, placing the clocks with value ∞ to the right: let $(i_1, \ldots, i_n)$ be a permutation of $(1, \ldots, n)$ such that, for all $1 \leq j < k \leq n$,

1. $v(x_{i_j}) = \infty \Rightarrow v(x_{i_k}) = \infty$, and
2. $v(x_{i_j}) \neq \infty \neq v(x_{i_k}) \Rightarrow \mathit{fract}(v(x_{i_j})) \leq \mathit{fract}(v(x_{i_k}))$.

From $v \cong v'$ and the definition of region equivalence it follows that properties (1) and (2) also hold if we replace each occurrence of v by v'. Using the properties of region equivalence, we infer that there exists a continuous, strongly monotone function $\rho' : [0, 1) \to [0, 1)$ with $\rho'(0) = 0$ and, for all j with $v(x_{i_j}) \neq \infty$, $\rho'(\mathit{fract}(v(x_{i_j}))) = \mathit{fract}(v'(x_{i_j}))$. We extend ρ' to a uniform mapping ρ with the required property by defining $\rho(\infty) = \infty$ and, for $t \in \mathsf{R}$, $\rho(t) \triangleq \lfloor t \rfloor + \rho'(\mathit{fract}(t))$. ⊠

Lemma 4.6. *$\overline{\rho(v)}(e) = \rho(\overline{v}(e))$.*

Lemma 4.7. *Whenever ρ is a uniform mapping then for every $d \in \mathsf{R}^{\geq 0}$ the mapping ρ_d, defined by $\rho_d(t) \triangleq \rho(t - d) - \rho(-d)$ for every $t \in \mathsf{R}^\infty$, is also uniform.*

[2]Čerāns [Čer92b, Čer92a] does not require uniform mappings to be continuous as he should have since his proof of Lemma 3.7 in [Čer92b] (which coincides with Lemma 11.7 in [Čer92a]) uses the property that a uniform mapping has an inverse that is also uniform.

Lemma 4.8. $\rho_d(v \oplus d) = \rho(v) \oplus -\rho(-d)$

Proof.

1. Assume x is a clock with $v(x) + d \in \mathit{intv}(x)$. Then

$$\begin{aligned}\rho_d(v \oplus d)(x) &= \rho_d((v \oplus d)(x)) = \rho_d(v(x) + d) = \rho(v(x) + d - d) - \rho(-d)\\ &= \rho(v)(x) - \rho(-d)\end{aligned}$$

2. Assume x is a clock with $v(x) + d \notin \mathit{intv}(x)$. In this case

$$\rho_d(v \oplus d)(x) = \rho_d((v \oplus d)(x)) = \rho_d(\infty) = \infty$$

3. We claim that $v(x) + d \in \mathit{intv}(x) \Leftrightarrow \rho(v)(x) - \rho(-d) \in \mathit{intv}(x)$. Using the uniformity of ρ and ρ^{-1}, we derive, for any integer n and $\Box \in \{<, \leq, >, \geq\}$,

$$v(x) + d \,\Box\, n \;\Leftrightarrow\; v(x) \,\Box\, n - d \;\Leftrightarrow\; \rho(v)(x) \,\Box\, n + \rho(-d) \;\Leftrightarrow\; \rho(v)(x) - \rho(-d) \,\Box\, n$$

Since $\mathit{intv}(x)$ has integer bounds, the claim follows from the combination of the derived inequalities.

The lemma now follows from (1), (2), (3) and the definition of $\oplus$. ⊠

The next two lemmas, which are the main results about uniform mappings, assert that uniform mappings "preserve" transitions between states.

Lemma 4.9. *If $s \xrightarrow{a} s'$ and $a \in \Sigma_{\mathcal{M}}$ then $\rho(s) \xrightarrow{a} \rho(s')$.*

Proof. Assume $s \xrightarrow{a} s'$ and ρ is a uniform mapping. Let $s = (q, v)$, $s' = (q', v')$, $\rho(v) = w$ and $\rho(v') = w'$. We must prove that $(q, w) \xrightarrow{a} (q', w')$.

Because $s \xrightarrow{a} s'$, there exists a transition δ such that $\delta : q \xrightarrow{a} q'$, $v \models G(\delta)$ and $v' = \overline{v} \circ A(\delta)$.
Since $w = \rho(v)$, Lemma 4.5 implies $v \cong w$. Hence, according to Lemma 4.2, $w \models G(\delta)$.
Assume that x is a clock. Then we derive, using the assumptions, definitions and Lemma 4.6,

$$\begin{aligned}w'(x) &= \rho(v')(x) = \rho(v'(x))\\ &= \rho(\overline{v} \circ A(\delta)(x)) = \rho(\overline{v}(A(\delta)(x))) = \overline{\rho(v)}(A(\delta)(x))\\ &= \overline{w}(A(\delta)(x)) = \overline{w} \circ A(\delta)(x)\end{aligned}$$

This means that $w' = \overline{w} \circ A(\delta)$. Combining this fact with $\delta : q \xrightarrow{a} q'$ and $w \models G(\delta)$, we may now conclude that $(q, w) \xrightarrow{a} (q', w')$, as required. ⊠

Lemma 4.10. *If $s \xrightarrow{d} s'$ then $\rho(s) \xrightarrow{-\rho(-d)} \rho_d(s')$.*

Proof. Assume $s \xrightarrow{d} s'$. Let $s = (q, v)$, $s' = (q, v')$, $\rho(v) = w$ and $\rho_d(v') = w'$. We must prove that $(q, w) \xrightarrow{-\rho(-d)} (q, w')$.

Since $s \xrightarrow{d} s'$, we know that $v' = v \oplus d$ and, for all $0 \leq d' \leq d$: $v \oplus d' \models \mathit{Inv}(q)$. By Lemma 4.8,

$$w' = \rho_d(v \oplus d) = \rho(v) \oplus -\rho(-d) = w \oplus -\rho(-d)$$

Choose $0 \leq d' \leq -\rho(-d)$. Let $d'' = -\rho^{-1}(-d')$. We derive

$$0 \leq d' \leq -\rho(-d) \;\Leftrightarrow\; 0 \geq -d' \geq \rho(-d) \;\Leftrightarrow\; 0 \geq \rho^{-1}(-d') \geq -d \;\Leftrightarrow\; 0 \leq d'' \leq d$$

This implies that $v \oplus d'' \models Inv(q)$. Since $\rho_{d''}$ is uniform (Lemma 4.7), uniform mappings preserve regions (Lemma 4.5), and regions preserve constraints (Lemma 4.2), $\rho_{d''}(v \oplus d'') \models Inv(q)$. By Lemma 4.8,

$$\rho_{d''}(v \oplus d'') \;=\; \rho(v) \oplus -\rho(-d'') \;=\; \rho(v) \oplus d'$$

Hence $\rho(v) \oplus d' \models Inv(q)$.

Since we have now proved that $w' = w \oplus -\rho(-d)$ and, for all $0 \leq d' \leq -\rho(-d)$, $\rho(v) \oplus d' \models Inv(q)$, it follows that $(q, w) \xrightarrow{-\rho(-d)} (q, w')$. ⊠

4.3 Grid Automata

After the preparatory subsections on regions and uniform mappings, we can now state and prove the key theorems that will enable us to restrict to finite subautomata when testing infinite timed transition systems. These subautomata will only contain states in which each clock value is either ∞ or in the *grid set* G^n, i.e., the set of integer multiples of 2^{-n}, for some sufficiently large natural number n.

For t a real number, let $\lfloor t \rfloor_n$ denote the largest number in G^n that is not greater than t, and let $\lceil t \rceil_n$ denote the smallest number in G^n that is not smaller than t. Write $[t]_n$ for the fraction $(\lfloor t \rfloor_n + \lceil t \rceil_n)/2$ (note that $[t]_n \in \mathsf{G}^{n+1}$). For $\mathcal{M}$ a TIOA, write $S^n_{\mathcal{M}}$ for the set of states $(q, v) \in S_{\mathcal{M}}$ such that, for each clock x, $v(x) \in \mathsf{G}^n \cup \{\infty\}$.

The two small technical lemmas below are easy to prove.

Lemma 4.11. *Let $s \in S^n_{\mathcal{M}}$.*

1. *If $s \xrightarrow{a} s'$ with $a \in \Sigma_{\mathcal{M}}$ then $s' \in S^n_{\mathcal{M}}$.*
2. *If $s \xrightarrow{d} s'$ with $d \in \mathsf{G}^n \cap \mathsf{R}^{>0}$ then $s' \in S^n_{\mathcal{M}}$.*

Lemma 4.12. *Let ρ be a uniform mapping, $u \in \mathsf{R}$ and $n \in \mathsf{N}$. Then there exists a uniform mapping ρ' such that, for all $t \in \mathsf{R}$,*

- *if $\rho(t) \in \mathsf{G}^n$ then $\rho'(t) = \rho(t)$,*
- *$\rho'(u) = [\rho(u)]_n$.*

The next theorem is an important step towards the discretization of state spaces. It asserts that whenever we have a distinguishing trace of length m for two states in $S^n_{\mathcal{M}}$, we can "massage" this trace into a trace in which all delay actions are in the grid set G^{n+m}.

Theorem 4.13. *Let $\mathcal{M}, \mathcal{M}'$ be TIOA's, let $(r, r') \cong (s, s')$ for states $r \in S_{\mathcal{M}}$, $r' \in S_{\mathcal{M}'}$, $s \in S^n_{\mathcal{M}}$ and $s' \in S^n_{\mathcal{M}'}$, and let $\sigma = a_1 a_2 \cdots a_m$ be a distinguishing trace for r and r'. Then there exists a distinguishing trace $\tau = b_1 b_2 \cdots b_m$ for s and s' such that, for all $j \in [1, \ldots, m]$, if a_j is an input or output action then $b_j = a_j$, and if a_j is a delay action then $b_j \in \mathsf{G}^{n+j}$ with $\lfloor a_j \rfloor \leq b_j \leq \lceil a_j \rceil$.*

Proof. Without loss of generality we may assume that r has the trace σ, r' does not, a_m is an output action, and r' has the trace $\sigma' = a_1 a_2 \cdots a_{m-1}$. Let $r_0 a_1 r_1 a_2 r_2 \cdots r_{m-1} a_m r_m$ be the unique execution fragment of $\mathcal{M}$ with $r = r_0$ and trace σ, and let $r'_0 a_1 r'_1 a_2 r'_2 \cdots r'_{m-2} a_{m-1} r'_{m-1}$ be the unique execution fragment of $\mathcal{M}'$ with $r' = r'_0$ and trace σ'. Inductively we define states $s_0, \ldots, s_{m-1}$ and $s'_0, \ldots, s'_{m-1}$, actions $b_1, \ldots, b_{m-1}$, and uniform mappings $\rho^0, \ldots, \rho^{m-1}$ such that, for all $j \in [0, \ldots, m-1]$, if $j > 0$ then b_j satisfies the conditions from the theorem, $\rho^j(r_j, r'_j) = (s_j, s'_j)$, $s_j \in S^{n+j}_{\mathcal{M}}$, $s'_j \in S^{n+j}_{\mathcal{M}'}$, and if $j < m-1$ then $s_j \xrightarrow{b_{j+1}} s_{j+1}$ and $s'_j \xrightarrow{b_{j+1}} s'_{j+1}$.

To start with, define $s_0 = s$ and $s'_0 = s'$. Since $(r, r') \cong (s, s')$ there exists, by Lemma 4.5, a uniform mapping ρ^0 with $\rho^0(r_0, r'_0) = (s_0, s'_0)$.

Now suppose that, for some $j \in [0, \ldots, m-2]$, $\rho^j(r_j, r'_j) = (s_j, s'_j)$, $s_j \in S^{n+j}_{\mathcal{M}}$ and $s'_j \in S^{n+j}_{\mathcal{M}'}$. We distinguish between two cases:

- a_{j+1} is an input or output action. Then define $s_{j+1} = \rho^j(r_{j+1})$, $s'_{j+1} = \rho^j(r'_{j+1})$, $b_{j+1} = a_{j+1}$, and $\rho^{j+1} = \rho^j$. Since $r_j \stackrel{a_{j+1}}{\rightarrow} r_{j+1}$ and $r'_j \stackrel{a_{j+1}}{\rightarrow} r'_{j+1}$, it follows by Lemma 4.9 that $s_j \stackrel{b_{j+1}}{\rightarrow} s_{j+1}$ and $s'_j \stackrel{b_{j+1}}{\rightarrow} s'_{j+1}$. By construction $\rho^{j+1}(r_{j+1}, r'_{j+1}) = (s_{j+1}, s'_{j+1})$. By Lemma 4.11.1 we may conclude that $s_{j+1} \in S^{n+j+1}_{\mathcal{M}}$ and $s'_{j+1} \in S^{n+j+1}_{\mathcal{M}'}$.
- $a_{j+1} = d$ is a delay action. By Lemma 4.12 there exists a uniform mapping ρ such that $\rho(r_j, r'_j) = (s_j, s'_j)$ and $\rho(-d) = [\rho^j(-d)]_{n+j} \in \mathsf{G}^{n+j+1}$. Define $\rho^{j+1} = \rho_d$. Then ρ^{j+1} is a uniform mapping by Lemma 4.7. Let $s_{j+1} = \rho^{j+1}(r_{j+1})$, $s'_{j+1} = \rho^{j+1}(r'_{j+1})$, and $b_{j+1} = -\rho(-d)$. Then Lemma 4.10 yields $s_j \stackrel{b_{j+1}}{\rightarrow} s_{j+1}$ and $s'_j \stackrel{b_{j+1}}{\rightarrow} s'_{j+1}$. Straightforward calculations give $\lfloor a_{j+1} \rfloor \leq b_{j+1} \leq \lceil a_{j+1} \rceil$. Moreover, by Lemma 4.11.2, we obtain $s_{j+1} \in S^{n+j+1}_{\mathcal{M}}$ and $s'_{j+1} \in S^{n+j+1}_{\mathcal{M}'}$.

Lemma 4.5 yields $r_{m-1} \cong s_{m-1}$ and $r'_{m-1} \cong s'_{m-1}$. Let $b_m = a_m$. Using Lemma 4.2, we infer that $s_{m-1} \stackrel{b_m}{\rightarrow}$ and not $s'_{m-1} \stackrel{b_m}{\rightarrow}$. This implies that $\tau = b_1 b_2 \cdots b_m$ is a distinguishing trace of s and s'. ⊠

Theorem 4.13 allows us to "massage" each distinguishing trace into one in which all delay actions are in a grid set, but there is a dependence between the length of the trace and the granularity of the grid: the longer the trace the finer the grid. This is due to the fact that the distinguishing power of a distinguishing trace for two states r and r' entirely depends on the regions traversed when applying σ to r and r', respectively. In certain cases even a tiny delay in σ may cause the traversal to a new region as we see in the next example.

Example 4.14. Consider the family of TIOA's defined as follows. For each $n \in \mathsf{N}$ we define the TIOA $\mathcal{M}_n$ with input action *in* and output action *out*. It has two clocks x and y, both with domain $[0,1] \cup \{\infty\}$. The set of locations is $\{q_j |\ 0 \leq j \leq n\} \cup \{p\}$ where q_n is the initial location and the initial valuation v_0 sets all clocks to 0. The transitions are annotated as follows:

$$\begin{array}{lll@{\qquad}l}
q_j \xrightarrow{in,(x>0 \wedge y<1),x:=0} q_{j-1} & \text{if } 0 < j \leq n & & q_0 \xrightarrow{in,(y<1)} p \\
q_j \xrightarrow{in,(x=0 \wedge y<1)} p & \text{if } 0 < j \leq n & & p \xrightarrow{in,\mathsf{T}} p \\
q_j \xrightarrow{out,(y=1)} p & \text{if } 0 \leq j \leq n & &
\end{array}$$

Finally, the invariant is defined by $Inv(q_j) = (y \leq 1)$ for all j, and $Inv(p) = \mathsf{T}$. Figure 2 depicts automaton $\mathcal{M}_n$.

Clearly, if $m < n$, the TIOA $\mathcal{M}_m$ is simply the TIOA $\mathcal{M}_n$ whose initial location is changed into q_m (up to removal of unreachable locations).

The TIOA $\mathcal{M}_n$ has the property that if within 1 time unit at most n inputs have occurred at different (non-zero) time points, an output action will be generated at time 1. If more inputs arrive in this interval, or the system is fed with two inputs at the same point in time, no output is generated. In fact, this property can be seen from the following observations. Suppose that the system $\mathcal{M}_n$ has just arrived to a location q_j, $0 < j \leq n$. Then it is in the region $(q_j,\ x = 0 < y < 1)$, $0 < j < n$, or $(q_n,\ x = y = 0)$. In any case, the input *in* would take the system to location p and hence it prevents the occurrence of the output *out*. Otherwise, by waiting a little while, the system moves to the successor region $(q_j,\ 0 < x < y < 1)$ (respectively, $(q_n,\ 0 < x = y < 1)$). At this point, the input action would take the system to the next location q_{j-1}, more precisely to the region $(q_{j-1},\ x = 0 < y < 1)$, and hence the output action would still be enabled. Notice that, while the system is at location q_0, performing "an extra" *in* also disables the occurrence of *out*.

It is easy to check that $\mathcal{M}_n$ and $\mathcal{M}_m$ behave differently for n and m different. Yet in order to observe this difference a grid size of at most $\frac{1}{2+\min(n,m)}$ is required. Note that this also means that the grid size depends on the number of states, not just on the number of clocks.

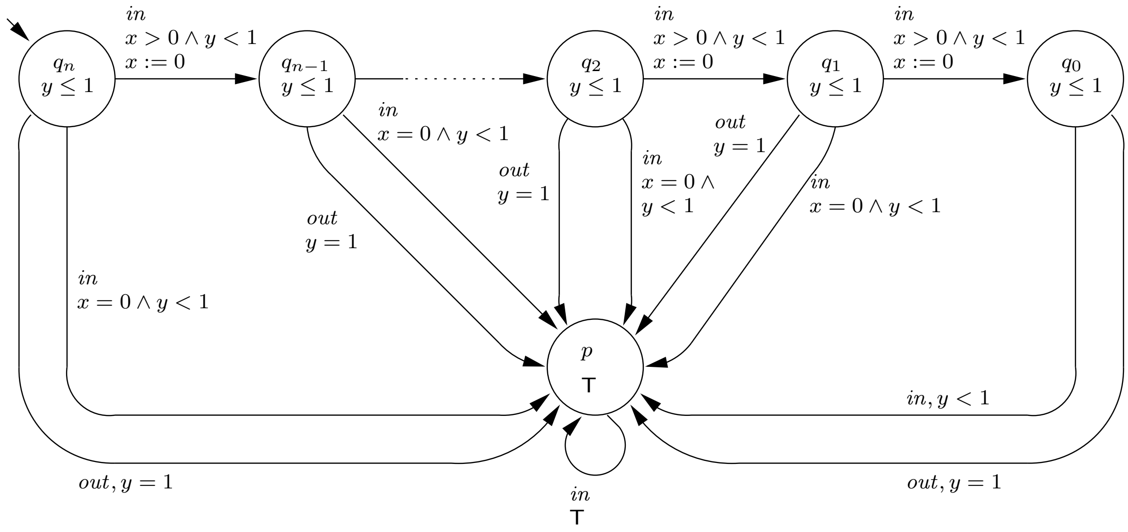

Figure 2: The TIOA $\mathcal{M}_n$

In order to obtain a grid size that is fine enough to distinguish all pairs of different states, we need to establish an upper bound on the length of minimal distinguishing traces. This is done in the following theorem.

Theorem 4.15. *Suppose $\mathcal{M}$ and $\mathcal{M}'$ are TIOA's with the same input actions, and r and s are states of $\mathcal{M}$ and $\mathcal{M}'$, respectively, with $\mathcal{OS}(\mathcal{M}), \mathcal{OS}(\mathcal{M}') : r \not\approx s$. Then there exists a distinguishing trace for r and s of length at most equal to the number of regions of $S_{\mathcal{M}} \times S_{\mathcal{M}'}$.*

Proof. Since r and s are not bisimilar there exists a trace that distinguishes between the two states. In fact, it is easy to see that there exists a distinguishing trace that ends with an output action. Among the distinguishing traces that end with an output action, let σ be a trace with minimal length. Assume that this length is greater than the number of regions of $S_{\mathcal{M}} \times S_{\mathcal{M}'}$. We derive a contradiction.

Assume, without loss of generality, that σ is a trace of r but not of s. Let

$$\begin{aligned} \beta &= r_0\, a_1\, r_1\, a_2\, \cdots\, a_{n-1}\, r_{n-1}\, a_n\, r_n \\ \gamma &= s_0\, a_1\, s_1\, a_2\, \cdots\, a_{n-1}\, s_{n-1} \end{aligned}$$

be the (uniquely determined) execution fragments of $\mathcal{M}$ and $\mathcal{M}'$, respectively, with $r = r_0$, $s = s_0$ and $\sigma = a_1 \cdots a_n$. Since n is greater than the number of regions of $S_{\mathcal{M}} \times S_{\mathcal{M}'}$, there exist indices $0 \leq i < j < n$ such that $(r_i, s_i) \cong (r_j, s_j)$. By Lemma 4.5, there exists a uniform mapping ρ such that $\rho(r_j, s_j) = (r_i, s_i)$. Repeated application of Lemmas 4.9 and 4.10 now allows us to construct a distinguishing trace for r_i and s_i of length $n - j$ that ends with an output action. But this means that there also exists a distinguishing trace for r and s of length $n + i - j$ that ends with an output action. Contradiction. ⊠

The upper bound on the length of distinguishing traces of Theorem 4.15 is of course astronomic in general. In specific cases, one can often give a much more reasonable upper bound. For instance, any pair of distinct states of the switch TIOA of Example 2.7 can be distinguished by a trace of length one (just wait long enough). In Example 2.8, any pair of inequivalent states of the TIOA associated to a Mealy machine can be distinguished by a trace with a length less than $2 \cdot \mid Q \mid$. (The factor 2 arises from the fact that we have split each transition in the Mealy machine into an input and an output part.) For each $\mathcal{M}_n$ in Example 4.14, any pair of different states can be distinguished by a trace of length at most $2n+1$: n input actions interleaved with $n+1$ appropriate delays.

For each TIOA $\mathcal{M}$ and natural number n, we define the *grid automaton* $\mathcal{G}(\mathcal{M}, n)$ as the subautomaton of $\mathcal{OS}(\mathcal{M})$ in which each clock value is in the set $\mathsf{G}^n \cup \{\infty\}$, and the only delay action is 2^{-n}. Note that since in the initial state of $\mathcal{OS}(\mathcal{M})$ all clocks take values in Z^∞, it is always included as a state of $\mathcal{G}(\mathcal{M}, n)$. Also observe that, since $\mathcal{G}(\mathcal{M}, n)$ is lean and has a finite number of states and actions, $\mathcal{G}(\mathcal{M}, n)$ is a finite automaton.

Definition 4.16. Let $\mathcal{M}$ be a TIOA and let $n \in \mathsf{N}$. The *grid automaton* $\mathcal{G}(\mathcal{M}, n)$ is the lean LTS $\mathcal{A}$ given by

- $Q_\mathcal{A} = S^n_\mathcal{M}$
- $\Sigma_\mathcal{A} = \Sigma_\mathcal{M} \cup \{2^{-n}\}$
- $q^0_\mathcal{A} = s^0_\mathcal{M}$
- for all $s, s' \in Q_\mathcal{A}$ and $a \in \Sigma_\mathcal{A}$, $s \xrightarrow{a}_\mathcal{A} s' \;\Leftrightarrow\; s \xrightarrow{a}_{\mathcal{OS}(\mathcal{M})} s'$

□

Example 4.17. The reader is invited to check that the picture of Figure 3 is the grid automaton $\mathcal{G}(\mathcal{M}_1, 1)$, where $\mathcal{M}_1$ is the TIOA from Example 4.14. To shorten notation, each state (q, v) has been denoted by $(q, v(x), v(y))$.

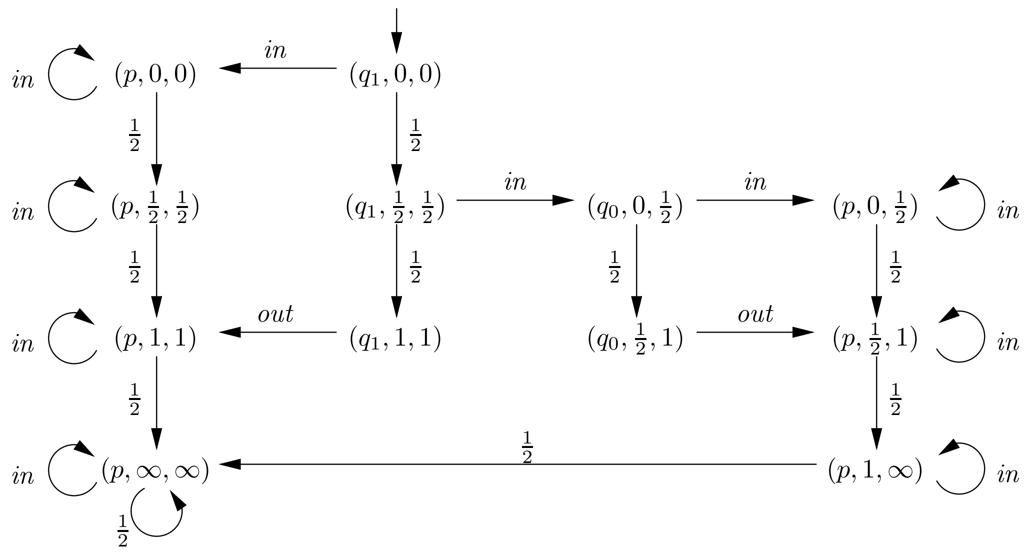

Figure 3: The grid automaton $\mathcal{G}(\mathcal{M}_1, 1)$

From the combination of Theorem 4.13 and Theorem 4.15, it now follows that two TIOA's are bisimilar if and only if their associated grid automata are bisimilar, provided the grid has been chosen sufficiently fine.

Corollary 4.18. Let $\mathcal{M}$ and $\mathcal{M}'$ be TIOA's with the same input actions, and let n be greater than or equal to the number of regions of $S_\mathcal{M} \times S_{\mathcal{M}'}$. Then $\mathcal{M} \simeq \mathcal{M}' \Leftrightarrow \mathcal{G}(\mathcal{M}, n) \simeq \mathcal{G}(\mathcal{M}', n)$.

Proof.
"$\Rightarrow$": Immediate, using Lemma 4.11, since $\mathcal{G}(\mathcal{M}, n)$ and $\mathcal{G}(\mathcal{M}', n)$ are proper subautomata of $\mathcal{M}$ and $\mathcal{M}'$, respectively.

"$\Leftarrow$": Suppose $\mathcal{M} \not\simeq \mathcal{M}'$. Then, by Theorem 4.15, there exists a trace σ of length at most n that distinguishes $s^0_{\mathcal{M}}$ and $s^0_{\mathcal{M}'}$. Since $s^0_{\mathcal{M}} \in S^0_{\mathcal{M}}$ and $s^0_{\mathcal{M}'} \in S^0_{\mathcal{M}'}$, application of Theorem 4.13 gives that there exists a trace τ that also distinguishes $s^0_{\mathcal{M}}$ from $s^0_{\mathcal{M}'}$ such that all delays in τ are elements of G^n. Let τ' be the trace obtained from τ by replacing each occurrence of a delay action $m \cdot 2^{-n}$, for m a positive natural number, by a sequence of m delay actions 2^{-n}. Using Wang's time additivity property (Lemma 2.6.2), we obtain that τ' also distinguishes $s^0_{\mathcal{M}}$ and $s^0_{\mathcal{M}'}$. Assume, w.l.o.g., that τ' is a trace of $s^0_{\mathcal{M}}$ but not of $s^0_{\mathcal{M}'}$. Since all the actions occurring in τ' are also actions of $\mathcal{G}(\mathcal{M}, n)$, it follows that τ' is a trace of $\mathcal{G}(\mathcal{M}, n)$ but not of $\mathcal{G}(\mathcal{M}', n)$. This contradicts $\mathcal{G}(\mathcal{M}, n) \simeq \mathcal{G}(\mathcal{M}', n)$. ⊠

We have now reduced the problem of deciding bisimulation equivalence of TIOA's to the problem of deciding bisimulation equivalence of two finite subautomata of the (highly infinite) operational semantics of these TIOA's. The main implication of this result for the conformance testing of TIOA's is that in the testing process we only need to explore finite subautomata, something that can be done effectively in finite time. Before we will address this issue in Section 5, we will prove as the final result of this section that if one applies a test sequence in which the only delay action is 2^{-n} to a TIOA $\mathcal{M}$, the resulting execution is fully contained in the grid automaton $\mathcal{G}(\mathcal{M}, n)$. This result (which is not entirely trivial) makes it possible to fully explore the grid subautomaton of a TIOA during the testing process. We need two small technical lemmas.

Lemma 4.19. *Suppose $v \models \mathit{hull}(\varphi)$. Then there exists at least one clock x with $v(x) \in \mathsf{Z}$.*

Proof. Suppose that for all clocks x, $v(x) \notin \mathsf{Z}$. Let $d = \frac{1}{2} \cdot (1 - \max_x \mathit{fract}(v(x)))$ and $v' = v \oplus d$. It is easy to check that $v \cong v'$. Therefore, by Lemma 4.2, $v' \models \mathit{hull}(\varphi)$. But since $d > 0$ this contradicts the assumption that v lies on the hull of φ. ⊠

Lemma 4.20. *Let $\mathcal{M}$ be a TIOA, $n \in \mathsf{N}$, and $s \in S^n_{\mathcal{M}}$. Suppose that $s \overset{2^{-n}}{\not\rightarrow}$. Then, for some output action $o \in O_{\mathcal{M}}$, $s \overset{o}{\rightarrow}$.*

Proof. Assume that s does not enable an output action. Then, by Lemma 2.6.3, s enables a delay action. However, as a consequence of Wang's additivity axiom (Lemma 2.6.2), all delay action that are enabled in s are less than 2^{-n}. Using Lemma 2.6.2 and 2.6.3 once more, we infer that there exists a delay $0 < d < 2^{-n}$, a state $s' = (q', v')$, and an output action o such that $s \overset{d}{\rightarrow} s' \overset{o}{\rightarrow}$. Clearly, in s' none of the clocks has a value in G^n. However, $s' \overset{o}{\rightarrow}$ implies that $v' \models \mathit{hull}(\mathit{Inv}(q'))$. By Lemma 4.19, this means that at least one clock has an integer value in v'. Contradiction. ⊠

For $\mathcal{M}$ a TIOA and $n \in \mathsf{N}$, write $\mathit{Exp}^n_{\mathcal{M}}$ for the set of test sequences of $\mathcal{M}$ in which all the delays are equal to 2^{-n}.

Lemma 4.21. *Let $\sigma \in \mathit{Exp}^n_{\mathcal{M}}$ and $s \in S^n_{\mathcal{M}}$. Then $\mathit{exec}_{\mathcal{M}}(\sigma, s)$ is an execution fragment of $\mathcal{G}(\mathcal{M}, n)$.*

Proof. By straightforward induction on the length of $\mathit{exec}_{\mathcal{M}}(\sigma, s)$. Use Lemma 4.20 to prove that rule (4) in Definition 3.1 can never be applied. ⊠

5 Deriving and Applying a Test Suite

Based on the results of Section 4, we introduce a test suite to test a timed implementation under test (IUT) for conformance with respect to a specification TIOA *Spec*. To prove that the test suite is indeed correct and complete relative to certain assumptions about the choice of parameters, we give in Figure 4 a simple test algorithm that applies each test case from the test suite to an implementation. Theorem 5.7 states that the algorithm is indeed correct. Our notion of conformance is as follows. We assume that the behavior of the IUT is accurately modeled by a TIOA *Impl*. Then the IUT conforms to the specification *Spec* if *Impl* is bisimilar to *Spec*. Note that we do not consider — as is often done — isomorphism between implementation and specification as conformance relation. This is due to the fact that we do not assume timed automata or their grid machines to be minimal.

Our method of building test suites is similar to Chow's classical algorithm for finite state Mealy machines [Cho78]. A test suite consists of a finite set of test sequences which should be applied to the implementation. Each sequence consists of the concatenation of two sequences. The initial part of a test sequence is taken from a *transition cover* P for a grid subautomaton of *Spec*, i.e., a set of test sequences that together exercise every transition of the subautomaton.

Definition 5.1. Let $\mathcal{M}$ be a TIOA, $n \in \mathsf{N}$, $\mathcal{A} = \mathcal{G}(\mathcal{M}, n)$. A *transition cover* for $\mathcal{A}$ is a finite collection $P \subseteq Exp^n_{\mathcal{M}}$ of test sequences, such that $\epsilon \in P$ and, for all transitions $s \xrightarrow{a} s'$ of $\mathcal{A}$ with s reachable (within $\mathcal{A}$) and stable, P contains test sequences σ and $\sigma\, a$ such that $s^0_{\mathcal{M}} \overset{\sigma}{\Rightarrow} s$. □

The trailing part of a test sequence is taken from a set Z, which is a *characterization set* for a grid subautomaton of *Impl*, meaning that for every pair of non-bisimilar grid states, Z contains a sequence that distinguishes between them.

Definition 5.2. Let $s \in S_{\mathcal{M}}$, $s' \in S_{\mathcal{M}'}$, and let σ be an experiment for $\mathcal{M}$ and $\mathcal{M}'$. We say that σ *distinguishes* s *from* s' if $outcome_{\mathcal{M}}(\sigma, s) \neq outcome_{\mathcal{M}'}(\sigma, s')$. If Z is a set of experiments for $\mathcal{M}$ and $\mathcal{M}'$, then we write $s \approx_Z s'$ if no experiment in Z distinguishes s from s'.

Let $n \in \mathsf{N}$. Then Z is a *characterization set for* $\mathcal{G}(\mathcal{M}, n)$ if, for all $s, s' \in S^n_{\mathcal{M}}$, $s \approx_Z s'$ implies $s \simeq_{\mathcal{G}(\mathcal{M},n)} s'$. □

To apply multiple test sequences to the IUT, we need, as in Chow's algorithm, the ability to always bring the machine back to its initial state. In the untimed setting of Mealy machines, one usually assumes the presence of a *reliable reset*, a special input action that brings the machine to its initial state from any given state. A similar requirement is needed in our timed setting, but in this case, it is not reasonable to consider the reset as an instantaneous operation: typically, some time will elapse between the moment when we request the machine to go to its initial state, and the moment at which the reset operation has been completed.[3] This motivates the following definition.

Definition 5.3. An input action a is a *reset* of a TIOA $\mathcal{M}$ if for each reachable, stable state s, $\mathcal{M}$ has an execution fragment of the form $s\ a\ \gamma$, where γ contains no input actions and has $s^0_{\mathcal{M}}$ as its last state. We say that $\mathcal{M}$ has *reset time* max if the maximal time that can elapse between the occurrence of reset a and the return to the initial state is at most max. □

It is not difficult to prove that the maximal time that can elapse between the occurrence of a reset action and the time at which the initial state is reached is always less than the number of regions of $\mathcal{M}$.

We find it convenient to further restrict attention to TIOA's with a *quiescent* initial state. In a quiescent state a machine waits for stimuli from its environment before producing any output. For instance, the switch of Example 2.7, in its initial state, always waits for the input action *on* and no output action can be (autonomously) generated. Similarly, the Mealy machine of Example 2.8

[3] Martin Wirsing came with the suggestive example of the control software on board of a BMW, in a state where the car races downhill with a speed of 200km/h.

is also quiescent in its initial state since in location q_0 only input actions are enabled. In contrast, the TIOA's $\mathcal{M}_n$ from Example 4.14 are not quiescent in their initial state since within 1 time unit the output action *out* is generated without any previous stimuli from the environment.

Definition 5.4. A state of a TIOA is *quiescent* if each outgoing execution fragment from that state that contains an output action also contains an input action. □

Now, we are in a position to formally define what a test suite for a given TIOA is.

Definition 5.5. Let $\mathcal{M}$ be a TIOA and $n \in \mathsf{N}$. Let P be a transition cover for $\mathcal{G}(\mathcal{M}, n)$ and Z a characterization set for the TIOA model of the IUT. The *test suite for $\mathcal{M}$ generated from P and Z with grid size n* is defined by

$$test.suite(\mathcal{M}, n, P, Z) \triangleq P\, Z\, \{\mathsf{reset\ max}\}$$

where concatenation of sets of sequences is as defined in Section 2.1. □

Figure 4 presents an algorithm that applies a test suite generated by our test method to an implementation. To prove its correctness, for appropriate values of its parameters, we need one more auxiliary lemma.

Input

1. A TIOA *Spec*, the *specification automaton*, with reset action reset, reset time max, and a quiescent initial state
2. An *Implementation Under Test (IUT)*, a device that accepts inputs from I_{Spec} and produces outputs in O_{Spec}
3. A natural number n
4. A natural number m

Output
A verdict *PASS* or *FAIL*

Algorithm
Let $X = I_{Spec} \cup \{2^{-n}\}$

1. Determine a (minimal) finite transition cover P for $\mathcal{G}(Spec, n)$
2. **For all** test sequences $\sigma \in test.suite(M, n, P, X^{m-1})$ **do**
   a. Apply test sequence σ to the IUT
   b. Return *FAIL* and halt if outcome of (a) differs from $outcome_{Spec}(\sigma, s^0_{Spec})$
3. Return *PASS* and halt

Figure 4: Application of test suite to TIOA's

Lemma 5.6. *Let $\mathcal{M}$ be a TIOA, $n \in \mathsf{N}$, and let m be greater than or equal to the number of states of $\mathcal{G}(\mathcal{M}, n)$. Let $Z = X^{m-1}$, where $X = I_{\mathcal{M}} \cup \{2^{-n}\}$. Then Z is a characterization set for $\mathcal{G}(\mathcal{M}, n)$.*

Proof. We prove that, for all states $s, s' \in S^n_{\mathcal{M}}$, $s \approx_Z s'$ implies $s \simeq_{\mathcal{G}(\mathcal{M},n)} s'$. Assume that $s \not\simeq_{\mathcal{G}(\mathcal{M},n)} s'$. Since $\mathcal{G}(\mathcal{M},n)$ is deterministic, there exists a distinguishing trace σ for s and s' of length $m-1$ or less [Koh78]. W.l.o.g. assume that $s \xrightarrow{\sigma\tau} r$, for some stable state r and sequence of output actions τ, and not $s' \xrightarrow{\sigma\tau}$. Then, by Lemma 3.3, $s \not\approx_Z s'$. ⊠

Like Chow, we need to give correct estimates of the size of the state spaces involved in order to obtain correctness of our method. Since, in general, the operational semantics of a TIOA has uncountably many states and transitions, measuring the state space of a TIOA gives no meaningful estimates. Instead, we provide estimates in terms of the number of regions of the product TIOA and the size of a grid subautomaton of the implementation.

The implications of the following theorem are twofold. It states that (a) the set $\mathit{test.suite}(\mathcal{M}, n, P, X^{m-1})$ is a complete test suite for appropriate m and n, and (b) the algorithm of Figure 4, when it takes such a test suite as an input, is correct in the sense that it returns *PASS* if and only if the IUT conforms to the given TIOA specification.

Theorem 5.7. *Let IUT and Spec be as in the algorithm of Figure 4. Assume that the behavior of IUT is accurately modeled by a TIOA Impl with reset action* reset*, reset time* max*, a quiescent initial state, and the same input and output actions as Spec. Assume that n is greater than or equal to the number of regions of $S_{Impl} \times S_{Spec}$, and m is greater than or equal to the number of states of $\mathcal{G}(Impl, n)$.*

Then the algorithm of Figure 4, when provided with these inputs, returns PASS iff Spec $\simeq$ Impl.

Proof. The *if* part is straightforward. As to the *only if* part, suppose that the algorithm returns *PASS*. By Corollary 4.18 it suffices to prove $\mathcal{G}(Spec, n) \simeq \mathcal{G}(Impl, n)$. Let P and X be defined as in the description of the algorithm. Define $Z = x^{m-1}$. Since IUT is accurately modeled by *Impl*, which has reset action reset and a quiescent initial state, and because the algorithm returns *PASS*, it follows that, for all $\sigma \in P$ and $\tau \in Z$,

$$\mathit{outcome}_{Spec}(\sigma\ \tau, s^0_{Spec}) \quad = \quad \mathit{outcome}_{Impl}(\sigma\ \tau, s^0_{Impl})$$

Let R be the relation between states given by

$$\begin{aligned} R \ = \ \{(s,r) \in S^n_{Spec} \times S^n_{Impl} \quad | \quad & \exists \sigma \in P\ \exists \beta_1, \beta_2, \gamma_1, \gamma_2 : \\ & \wedge\ \mathit{exec}_{Spec}(\sigma, s^0_{Spec}) = \beta_1\ \beta_2 \\ & \wedge\ \mathit{exec}_{Impl}(\sigma, s^0_{Impl}) = \gamma_1\ \gamma_2 \\ & \wedge\ s = \mathit{last}(\beta_1) \\ & \wedge\ r = \mathit{last}(\gamma_1) \\ & \wedge\ \mathit{trace}(\beta_1) = \mathit{trace}(\gamma_1) \\ & \wedge\ \mathit{trace}(\beta_2) = \mathit{trace}(\gamma_2) \in (O_{Spec})^* \} \end{aligned}$$

Note that $s\ R\ r$ implies $s \approx_Z r$. Write $\mathcal{A} = \mathcal{G}(Impl, n)$. We claim that the relation $R' = R \circ \simeq_{\mathcal{A}}$ is a bisimulation between $\mathcal{G}(Spec, n)$ and $\mathcal{G}(Impl, n)$.

Since $\epsilon \in P$ and $\epsilon \in Z$, we obtain $\mathit{outcome}_{Spec}(\epsilon, s^0_{Spec}) = \mathit{outcome}_{Impl}(\epsilon, s^0_{Impl})$. This implies that $s^0_{Spec}\ R\ s^0_{Impl}$. Hence, since $\simeq_{\mathcal{A}}$ is reflexive, $s^0_{Spec}\ R'\ s^0_{Impl}$.

For the proof of the transfer property, assume that $s\ R'\ r$ and $s \xrightarrow{a} s'$. Then there exists a state u such that $s\ R\ u \simeq_{\mathcal{A}} r$. We distinguish between two cases:

1. a is an output action. From the definition of R in combination with Lemma 2.6.3 it follows that $u \xrightarrow{a} u'$, for some state u' with $s'\ R\ u'$. Since $\simeq_{\mathcal{A}}$ is a bisimulation, there exists a state r' such that $r \xrightarrow{a} r'$ and $u' \simeq_{\mathcal{A}} r'$. Then $s'\ R'\ r'$, as required.

2. a is an input or delay action. Then it follows from the definition of R in combination with Lemma 2.6.3 that both s and u are stable states. By definition of R, there exists an experiment $\sigma \in P$ such that $s^0_{Spec} \stackrel{\sigma}{\Rightarrow} s$ and $s^0_{Impl} \stackrel{\sigma}{\Rightarrow} u$. This implies in particular that s is a reachable state, and therefore, since P is a transition cover, P contains test sequences τ and $\tau\, a$ such that $s^0_{Spec} \stackrel{\tau}{\Rightarrow} s$. Let $u'' = \mathit{last}(\mathit{exec}_{Impl}(\tau, s^0_{Impl}))$. Then $s \; R \; u''$. Since $s \approx_Z u$, $s \approx_Z u''$ and $\approx_Z$ is an equivalence relation, $u \approx_Z u''$. By Lemma 5.6, this implies $u \simeq_{\mathcal{A}} u''$. Since also $u \simeq_{\mathcal{A}} r$ and $\simeq_{\mathcal{A}}$ is an equivalence relation, this implies $u'' \simeq_{\mathcal{A}} r$. Let u' be the unique state satisfying $u'' \xrightarrow{a} u'$. Using that $\mathit{outcome}_{Spec}(\tau\ a, s^0_{Spec}) = \mathit{outcome}_{Impl}(\tau\ a, s^0_{Impl})$, we infer $s' \; R \; u'$. Now we use the fact that $u'' \simeq_{\mathcal{A}} r$ to infer that there exists a state r' such that $r \xrightarrow{a} r'$ and $u' \simeq_{\mathcal{A}} r'$. Then $s' \; R' \; r'$, as required.

The other transfer property can be proved similarly. ⊠

6 Discussion

The algorithm presented in the previous section results in an astronomically large number of test sequences. On top of that, the time delays that occur in these test sequences are microscopically small. Clearly, our algorithm cannot be claimed to be itself of practical value. Rather, the major contribution of our paper is the TIOA model and the demonstration that an algorithm to derive a (complete!) test suite at least exists. In this section we discuss ways to reduce the number of tests, and to make the time delays within the tests manageable.

We have deliberately tried to impose as few restrictions on the model as possible and, as a consequence, our model is extremely fine-grained. It is for instance possible to model situations where occurrences of an input action at two distinct but arbitrarily close moments (see Example 4.14) lead to completely different behavior. Obviously, much of this subtlety can be sacrificed while retaining a sufficiently expressive model. Finding suitable special cases of our model is therefore an urgent issue. The first two authors are currently preparing a paper on *Bounded Response Automata*, a special case of TIOA's which appears to be sufficiently expressive for most applications but can nevertheless be fully tested by considering grid subautomata with time steps of size 1. Another possibility in this direction is to obtain more *robust* versions (in the sense of [GHJ97]) of our timed I/O automata. Basically, a TIOA is robust if, whenever it accepts a trace, it accepts a "neighboring" trace as well, under some reasonable topology on the set of traces. We hope that a more robust model will yield significantly smaller test sets.

An alternative line of attack is to optimize on the granularity of the grid. In our approach, the granularity of the grid is directly derived from an upper bound on the length of distinguishing traces: the shorter the distinguishing traces, the coarser the grid. So in order for our approach to become practical, it is vital to derive good upper bounds on the length of distinguishing traces. We hope that modifications of algorithms for deciding bisimulation equivalence (such as presented in [Čer92b, Čer92a, WL97]) can yield such bounds. These algorithms might also be helpful to improve on the construction of distinguishing sequences.

A remarkable property of our method is that, unlike most testing approaches for Mealy machines (with some exceptions, e.g., [PHK95]), we do not assume minimality of the specification and implementation automata. Nevertheless, for reasons of efficiency it is of course desirable to work with minimal automata. Minimization of timed automata, however, is a non-trivial issue; in particular timed automata in the Alur-Dill model [AD94] (and BTDA's) cannot be minimized in general. To solve this problem, in [SV96] the *minimizable timed automata* (MTA) model is introduced as an extension of the BTDA model. This model does allow minimization: for every MTA there exists a minimal MTA with bisimilar operational semantics. We hope that by working with minimal timed automata the size of test sets can be further reduced by using, for instance, techniques for discrete time automata, like in [CG98].

Finally, we expect that our approach may also support incomplete but practically useful methods for testing timed systems such as in [CL97, MMM95]. In fact, it is always an option to use the grid automaton construction heuristically. Instead of taking the worst-case grid size right away, one might start off with a coarse grid to obtain a small, incomplete set of useful tests. If desired, the grid can be subsequently refined, thus approximating the required grid size. After all, the bound for the grid size that we require is indeed very small, and we hope that there is room for some improvement. In fact, we do not know of any counterexample that states that such a bound is tight. Example 4.14 only points out that for general TIOA's the grid size can never be based on the number of clocks alone but it also suggests that a still appropriate grid size might be far above the bound we obtained.

References

[AD94] R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[ADLU91] A.V. Aho, A.T. Dahbura, D. Lee, and M.Ü. Uyar. An optimization technique for protocol conformance test generation based on UIO sequences and Rural Chinese Postman Tours. *IEEE Transactions on Communications*, 39(11):1604–1615, 1991.

[AH96] R. Alur and T.A. Henzinger, editors. *Proceedings of the 8th International Conference on Computer Aided Verification,* New Brunswick, NJ, USA, volume 1102 of *Lecture Notes in Computer Science*. Springer-Verlag, July/August 1996.

[AHS96] R. Alur, T.A. Henzinger, and E.D. Sontag, editors. *Hybrid Systems III*, volume 1066 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[AK96] R. Alur and R.P. Kurshan. Timing analysis in COSPAN. In Alur et al. [AHS96], pages 220–231.

[BGK+96] J. Bengtsson, W.O.D. Griffioen, K.J. Kristoffersen, K.G. Larsen, F. Larsson, P. Pettersson, and Wang Yi. Verification of an audio protocol with bus collision using UPPAAL. In Alur and Henzinger [AH96], pages 244–256.

[CG98] R. Cardell-Oliver and T. Glover. A practical and complete algorithm for testing real-time systems. Technical Report CSM-306, Department of Computer Science, University of Essex, February 1998.

[Čer92a] K. Čerāns. *Algorithmic Problems in Analysis of Real Time System Specifications.* Dr.sc.comp. thesis, University of Latvia, Rīga, 1992.

[Čer92b] K. Čerāns. Decidability of bisimulation equivalences for parallel timer processes. In G. v. Bochmann and D.K. Probst, editors, *Proceedings of the 4th International Workshop on Computer Aided Verification,* Montreal, Canada, volume 663 of *Lecture Notes in Computer Science*, pages 302–315. Springer-Verlag, 1992.

[Cho78] T.S. Chow. Testing software design modeled by finite-state machines. *IEEE Transactions on Software Engineering*, 4(3):178–187, 1978.

[CL97] D. Clarke and I. Lee. Automatic generation of tests for timing constraints from requirements. In *Proceedings of the Third International Workshop on Object-Oriented Real-Time Dependable Systems,* Newport Beach, California, February 1997.

[CVI89] W.Y.L. Chan, S.T. Vuong, and M.R. Ito. An improved protocol test generation procedure based on UIOs. In *Proceedings of the ACM Symposium on Communication Architectures and Protocols*, pages 283–294. ACM, 1989.

[DKRT97] P.R. D'Argenio, J.-P. Katoen, T.C. Ruys, and J. Tretmans. The bounded retransmission protocol must be on time! In E. Brinksma, editor, *Proceedings of the Third Workshop on Tools and Algorithms for the Construction and Analysis of Systems,* Enschede, The Netherlands, volume 1217 of *Lecture Notes in Computer Science*, pages 416–431. Springer-Verlag, April 1997.

[DOTY96] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool KRONOS. In Alur et al. [AHS96], pages 208–219.

[DY95] C. Daws and S. Yovine. Two examples of verification of multirate timed automata with KRONOS. In *Proceedings of the 16th IEEE Real-Time Systems Symposium (RTSS'95),* Pisa, Italy, pages 66–75. IEEE Computer Society Press, December 1995.

[FJJV96] J.-C. Fernandez, C. Jard, Th. Jéron, and C. Viho. Using on-the-fly verification techniques for the generation of test suites. In Alur and Henzinger [AH96], pages 348–359.

[GHJ97] V. Gupta, T.A. Henzinger, and R. Jagadeesan. Robust timed automata. In O. Maler, editor, *Proceedings International Workshop HART'97*, volume 1201 of *Lecture Notes in Computer Science*, pages 331–345. Springer-Verlag, 1997.

[HH95] T.A. Henzinger and P.-H. Ho. HyTech: The Cornell HYbrid TECHnology Tool. In U.H. Engberg, K.G. Larsen, and A. Skou, editors, *Proceedings of the Workshop on Tools and Algorithms for the Construction and Analysis of Systems,* Aarhus, Denmark, volume NS-95-2 of *BRICS Notes Series*, pages 29–43. Department of Computer Science, University of Aarhus, May 1995.

[Hol91] G.J. Holzmann. *Design and Validation of Computer Protocols.* Prentice-Hall International, 1991.

[HWT95] P.-H. Ho and H. Wong-Toi. Automated analysis of an audio control protocol. In P. Wolper, editor, *Proceedings of the 7th International Conference on Computer Aided Verification,* Liège, Belgium, volume 939 of *Lecture Notes in Computer Science*. Springer-Verlag, June 1995.

[Koh78] Z. Kohavi. *Switching and Finite Automata Theory.* McGraw-Hill, Inc., Second edition, 1978.

[LPY97] K.G. Larsen, P. Pettersson, and Wang Yi. UPPAAL in a nutshell. *International Journal on Software Tools and Technology Transfer*, 1(1/2):134–152.

[LT87] N.A. Lynch and M.R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proceedings of the* 6^{th} *Annual ACM Symposium on Principles of Distributed Computing*, pages 137–151, August 1987. A full version is available as MIT Technical Report MIT/LCS/TR-387.

[MMM95] D. Mandrioli, S. Morasca, and A. Morzenti. Generating test cases for real-time systems from logic specifications. *ACM Transactions on Computer Systems*, 13(4):365–398, 1995.

[PHK95] A. Petrenko, T. Higashino, and T. Kaji. Handling redundant and additional states in protocol testing. In A. Cavalli and S. Budkowski, editors, *Proceedings of the 8th International Workshop on Protocol Test Systems IWPTS '95,* Paris, France, pages 307–322, 1995.

[SGSL98] R. Segala, R. Gawlick, J.F. Søgaard-Andersen, and N. Lynch. Liveness in timed and untimed systems. *Information and Computation*, 141:119–171, 1998.

[Som96] I. Sommerville. *Software Engineering, 5/e.* Addison-Wesley Publishing Company, 1996.

[SV96] J.G. Springintveld and F.W. Vaandrager. Minimizable timed automata. In B. Jonsson and J. Parrow, editors, *Proceedings of the Fourth International Symposium on Formal Techniques in Real Time and Fault Tolerant Systems (FTRTFT'96),* Uppsala, Sweden, volume 1135 of *Lecture Notes in Computer Science*, pages 130–147. Springer-Verlag, 1996.

[Tre92] J. Tretmans. *A Formal Approach to Conformance Testing*. PhD thesis, University of Twente, December 1992.

[Tre96a] J. Tretmans. Test generation with inputs, outputs, and quiescence. In T. Margaria and B. Steffen, editors, *Proceedings of the Workshop on Tools and Algorithms for the Construction and Analysis of Systems,* Passau, Germany, volume 1055 of *Lecture Notes in Computer Science*, pages 127–146. Springer-Verlag, April 1996.

[Tre96b] J. Tretmans. Test generation with inputs, outputs, and repetitive quiescence. *Software–Concepts and Tools*, 17:103–120, 1996.

[WL97] C. Weise and D. Lenzkes. Efficient scaling-invariant checking of timed bisimulation. In R. Reischuk and M. Morvan, editors, *Proceedings of the 14th Symposium on Theoretical Aspects of Computer Science STACS '97,* Lübeck, Germany, volume 1200 of *Lecture Notes in Computer Science*, pages 177–188. Springer-Verlag, February 1997.

[Yi90] Wang Yi. Real-time behaviour of asynchronous agents. In J.C.M. Baeten and J.W. Klop, editors, *Proceedings CONCUR 90,* Amsterdam, volume 458 of *Lecture Notes in Computer Science*, pages 502–520. Springer-Verlag, 1990.

A Notational Conventions

a	action
d	nonnegative real number
e	term
i	input action, index
j, k	index
m, n	integer or ∞
o	output action
q, r, s	state or location
t, u	real number or ∞
v, w	clock valuation
x, y, z	clock
A	mapping from transitions to assignments
C	finite set of clocks
E	set of transitions
$F(C)$	the set of constraints over C
G	mapping from transitions to constraints
I	set of input actions
J	interval
$M(C)$	the set of assignments over C
O	set of output actions
P	transition cover
Q	set of states or locations
R	relation
S	set of states
$T(C)$	the set of terms over C
$V(C)$	the set of clock valuations over C
X	set of actions
Z	set of sequences of actions
$\mathcal{A}$	labeled transition system
$\mathcal{B}$	bounded time domain automaton
$\mathcal{E}$	experiment LTS
$\mathcal{G}$	grid automaton
$\mathcal{M}$	timed I/O automaton
$\mathcal{OS}(\mathcal{B})$	the operational semantics of $\mathcal{B}$
$\mathcal{P}$	input/output partition of action set
$\mathcal{T}$	timing annotation
F	falsehood
G^n	the set of integer multiples of 2^{-n}
N	the set of natural numbers
R	the set of real numbers
T	truth
Z	the set of integers
α	region
β, γ	execution fragment
δ	transition
ϵ	the empty sequence
μ	assignment
ρ	uniform mapping
σ, τ	sequence
φ	constraint
Σ	set of actions