Specifying Urgency in Timed I/O Automata[*]

Biniam Gebremichael Frits Vaandrager
Institute for Computing and Information Sciences
Radboud University Nijmegen, The Netherlands
{B.Gebremichael,F.Vaandrager}@cs.ru.nl

Abstract

Tools and techniques based on timed automata (such as Uppaal and the timed I/O automata framework) have proven to be extremely useful for the analysis of protocols and control software for real-time systems. However, a significant limitation of these approaches is that, due to the expressiveness of the modeling languages, timelocks — degenerate states in which time is unable to pass — can freely arise and cannot, in the general case, be detected. As a remedy to this problem Sifakis et al. advocate the use of deadline *predicates for the specification of progress properties of Alur-Dill style timed automata. In this article, we extend these ideas to a more general setting, which may serve as a basis for deductive verification techniques. More specifically, we extend the TIOA framework of Lynch et al with* urgency *predicates. We identify a suitable language to describe the resulting* timed I/O automata with urgency *and show that for this language time reactivity holds by construction. We also establish that the class of timed I/O automata with urgency is closed under composition. The use of urgency predicates is compared with three alternative approaches to specifying progress properties that have been advocated in the literature: invariants, stopping conditions and deadline predicates. We argue that in practice the use of urgency predicates leads to shorter and more natural specifications than any of the other approaches. Some preliminary results on proving invariant properties of timed (I/O) automata with urgency are presented.*

1. Introduction

In the literature on real-time systems there appears to be broad consensus on how to express quantitative timing constraints in state based modeling formalisms. Following the approach advocated by Alur and Dill [1], the idea is to designate certain state variables as clock variables. The values of these clock variables change as time advances. Also, clocks may be reset when discrete events occur. Timing constraints can be expressed, then, by conditions on clock values.

One issue on which there is no general consensus yet is how to specify progress properties, that is, properties which assert that a system must perform a certain action before a certain point in time. Merritt et al. in [22] propose a model with upper and lower bounds associated with tasks (that is, sets of system actions). In the work of Alur and Dill [1], progress is enforced via a Büchi style acceptance criterion: by requiring that some (sets of) locations are visited infinitely often the possibility is ruled out that a system stays in certain locations forever. A popular approach, which is advocated in [14, 2] and implemented in the tool UPPAAL [18], is to use (state) invariants. An invariant typically enforces a system action by limiting the amount by which time may advance in a given state. A related approach that is pursued in [16] is to use stopping conditions. Here the idea is that when a system reaches a state in which a stopping condition holds, time may not progress any further and a system action has to occur immediately. Sifakis and his colleagues [4, 24] advocate the use of deadlines for the specification of progress properties. Each transition of an Alur-Dill style timed automaton is decorated with an additional deadline predicate, which specifies when the transition becomes urgent. An advantage of deadline approach (which can be viewed as a generalization of the approach of [22]) is that under some reasonable assumptions, it ensures what is called time reactivity in [4] and timelock freedom in [5], that is, whenever time progress stops there exists at least one enabled transition. Under certain conditions, time reactivity is even preserved by parallel composition of automata [5, 4, 3]. The notion of deadlines has been incorporated in several modeling frameworks, see for instance [5, 13], and it has been implemented as part of the IF toolset [6] and MoDeST [8].

The work of Sifakis et al [24, 4] takes place in a setting

[*] This work was supported by the European Community Project IST-2001-35304 Advanced Methods for Timed Systems (AMETIST), http://ametist.cs.utwente.nl.

of Alur-Dill style timed automata, a system model that has limited expressivity in order to enable automatic state space exploration and model checking. In this article, we study the specification of progress properties in the much more general model of timed I/O automata (TIOA) of Lynch et al [16]. Even though fragments of the TIOA framework can be translated into timed automata [23], analysis of general TIOA models requires the use of deductive verification techniques and theorem provers such as PVS [15]. Inspired by the work of Sifakis et al, we introduce a similar notion of *urgency* predicates within the TIOA framework, both at the semantic level where we have infinite sets of states, transitions and trajectories, and at the syntactic level where system behavior is described finitely in terms of a logical language.

In the I/O automaton framework, transitions are typically specified using precondition/effect notation, that is, some type of guarded commands. This means that, for a given action name b with parameters $\vec{h}$ a precondition predicate $pre(\vec{v}, \vec{h})$ is given that defines from which states $\vec{v}$ action $b(\vec{h})$ is enabled, and an effect predicate $\mathit{eff}(\vec{v}, \vec{h}, \vec{v}')$ that defines to which states $\vec{v}'$ one may jump after doing action $b(\vec{h})$ in state $\vec{v}$. For the specification of timed systems we add a third predicate, the *urgency* predicate $urg(\vec{v}, \vec{h})$, to every transition definition. The meaning of the urgency predicate is that if, for some $\vec{h}$, the state predicate

$$pre(\vec{v}, \vec{h}) \wedge urg(\vec{v}, \vec{h}) \qquad (1)$$

becomes true at a time point t in a trajectory, then t must be the limit time of that trajectory. Intuitively, the precondition specifies when a transition *may* occur, and the urgency predicate specifies when the transition becomes urgent, that is, either this or some other enabled discrete transition *must* occur immediately. A small but significant difference between our approach and the one of Sifakis et al [24, 4] is that Sifakis et al require that a deadline predicate *implies* the precondition predicate, whereas we achieve a similar effect by *conjoining* the urgency predicate with the precondition.

The main contributions of this article are:

1. Extension of the work of [24, 4, 3] on deadline predicates to a much more expressive setting, which may serve as a basis for deductive verification techniques. More specifically, we extend the TIOA framework of [16] with urgency predicates at the semantic level, and define a suitable language to describe the resulting *timed I/O automata with urgency*. For this language, time reactivity holds by construction. We also establish that the class of TIOAs with urgency is closed under composition. In general, under the usual semantics, timed automata with urgency and timed automata with deadlines are not closed under composition, this problem has been studied in [7], where an alternative semantics is given that preserves compositionality.
2. A comparison of urgency predicates with three alternative ways to specifying progress properties: invariants [14, 2], stopping conditions [16] and deadlines [24, 4, 3]. Deadlines, stopping conditions and urgency predicates are shown to be (essentially) equally expressive. Invariants are slightly more expressive since they allow to bound the time at which an action occurs by a right open interval. Only use of urgency and deadline predicates gives time reactivity by construction. We argue that in practice the use of urgency predicates leads to shorter and more natural specifications than any of the other methods.
3. Some preliminary results on proving invariant properties of timed (I/O) automata with urgency. A similar approach for discrete time can also be found in [11].

The full version of the present paper appears as [10]. The proofs which have been omitted here, due to space limitation, are available in this technical report.

2. Timed (I/O) Automata with Urgency

In this section, we describe our extension of the timed I/O automata framework of Lynch et al [16, 17] with urgency. In Subsections 2.1 and 2.2 we begin with recalling some definitions from [16, 17]: we introduce a basic vocabulary for describing timed behaviors and recall the notion of a timed automaton. In Subsection 2.3, we add a notion of urgent transitions to timed automata, both at the semantic and at the syntactic level. The class of timed automata with urgency is not closed under composition, in general. In order to obtain compositionality, we add, in Subsection 2.4, an input/output distinction. Subsection 2.5, finally, defines a parallel composition operator and establishes that both the class of timed automata and the class of timed I/O automata with urgency are closed under composition.

2.1. Describing Timed System Behavior

In this section, we list the basic notions that are used in describing the behavior of a timed system, including both discrete and continuous changes. We simply sketch this material, leaving the reader to consult [16, 17] for the details.

The time domains we use is the set R of real numbers (in [16, 17] also other time domains are considered). States of automata will consist of valuations of *variables*. Each variable has both a *static type*, which defines the set of values it may assume, and a *dynamic type*, which gives the set of trajectories it may follow. We assume that dynamic types are closed under some simple operations: shifting the time domain, taking subintervals and pasting together intervals. We call a variable *discrete* if its dynamic type equals the pasting-closure of a set of constant-valued functions (i.e., the step-functions), and *analog* if its dynamic type equals the pasting-closure of a set of continuous functions (i.e., the piecewise-continuous functions).

A *valuation* for a set V of variables is a function that associates with each variable $v \in V$ a value in its static type. We write $val(V)$ for the set of all valuations for V. A *trajectory* for a set V of variables describes the evolution of the variables in V over time; formally, it is a function from a time interval that starts with 0 to valuations of V, that is, a trajectory defines a value for each variable at each time in the interval. We write $dom(\tau)$ for the domain of trajectory τ. A *point trajectory* is one with the trivial domain $\{0\}$. We write $\wp(\mathbf{x})$ for the point trajectory for valuation $\mathbf{x}$. The *limit time* of a trajectory τ, $\tau.ltime$, is the supremum of the times in its domain. $\tau.fval$ is defined to be the first valuation of τ, and if τ is right-closed, $\tau.lval$ is the last valuation. Suppose τ and τ' are trajectories for V, with τ closed. The *concatenation* of τ and τ', denoted by $\tau \frown \tau'$, is the trajectory obtained by taking the union of the first trajectory and the function obtained by shifting the domain of the second trajectory until the start time agrees with the limit time of the first trajectory; the last valuation of the first trajectory, which may not be the same as the first valuation of the second trajectory, is the one that appears in the concatenation. Trajectory τ is a prefix of trajectory τ', denoted $\tau \leq \tau'$, if τ can be obtained by restricting τ' to a subset of its domain. For every $t \in dom(\tau)$, we define $\tau \trianglerighteq t$ to be the trajectory obtained by taking the part of τ from t onwards, and then shifting the domain so that it starts with 0 again. Formally, $dom(\tau \trianglerighteq t) = \{u \in \mathsf{R} \mid u + t \in dom(\tau)\}$ and for all u in the domain, $\tau \trianglerighteq t(u) = \tau(u + t)$.

2.2. Timed Automata

A timed automaton in the sense of [16, 17] is a state machine whose states are divided into *variables* and that has a set of discrete *actions*. The state of a timed automaton may change in two ways: by *discrete transitions*, which change the state atomically, and by *trajectories*, which describe the evolution of the state over intervals of time. Discrete transitions are labeled with actions, which are classified as either *external* or *internal*. The external actions are used to synchronize with the automaton's environment, while the internal actions are only visible to the automaton itself.

Formally, a *timed automaton* is a tuple $\mathcal{A} = (X, Q, \Theta, E, H, \mathcal{D}, \mathcal{T})$ with

- A set X of *internal variables*.
- A set $Q \subseteq val(X)$ of *states*.
- A nonempty set $\Theta \subseteq Q$ of *start states*.
- A set E of *external actions* and a set H of *internal actions*, disjoint from each other. We write $A \triangleq E \cup H$.
- A set $\mathcal{D} \subseteq Q \times A \times Q$ of *discrete transitions*. An edge $e = (\mathbf{x}, a, \mathbf{x}') \in \mathcal{D}$, also written as $\mathbf{x} \xrightarrow{a} \mathbf{x}'$, represent a transition from state $\mathbf{x}$ to $\mathbf{x}'$ labeled with action a. We say that a is *enabled* in $\mathbf{x}$ if $\mathbf{x} \xrightarrow{a} \mathbf{x}'$ for some $\mathbf{x}'$.
- A set $\mathcal{T}$ of trajectories for X such that $\tau(t) \in Q$ for each $\tau \in \mathcal{T}$ and $t \in dom(\tau)$. We require that the following axioms hold:

  T0 *(Existence of point trajectories)* If $\mathbf{x} \in Q$ then $\wp(\mathbf{x}) \in \mathcal{T}$.

  T1 *(Prefix closure)* For every $\tau \in \mathcal{T}$ and every $\tau' \leq \tau, \tau' \in \mathcal{T}$.

  T2 *(Suffix closure)* For every $\tau \in \mathcal{T}$ and every $t \in dom(\tau), \tau \trianglerighteq t \in \mathcal{T}$.

  T3 *(Concatenation closure)* Let $\tau_0 \tau_1 \tau_2 \ldots$ be a sequence of trajectories in $\mathcal{T}$ such that, for each non final index, i, τ_i is closed and $\tau_i.lval = \tau_{i+1}.fval$. Then $\tau_0 \frown \tau_1 \frown \tau_2 \ldots \in \mathcal{T}$.

A trajectory τ is *maximal* in $\mathcal{T}$ if there exists no $\tau' \in \mathcal{T}$ with $\tau < \tau'$. The following lemma (which as far as we know is new) states that each trajectory can be extended into a maximal one. Intuitively this is an obvious property, but the proof requires some work due to the fact that we know so little about $\mathcal{T}$.

Lemma 1 *Let $\mathcal{A}$ be a timed automaton and let $\mathcal{T}$ be its set of trajectories. Then each trajectory in $\mathcal{T}$ is a prefix of a trajectory that is maximal in $\mathcal{T}$.*

2.3. Adding Urgency

We now extend timed automata with extra state predicates, *urgency predicates*, one for each action.

A *timed automaton with urgency* is a pair $(\mathcal{A}, U)$ of a timed automaton $\mathcal{A} = (X, Q, \Theta, E, H, \mathcal{D}, \mathcal{T})$ and an *urgency predicate* $U : Q \times A \rightarrow Bool$. If $U(\mathbf{x}, a) = \texttt{true}$ then we say that action a is *potentially urgent* in a state $\mathbf{x}$. Action a is *urgent* in state $\mathbf{x}$ if it is potentially urgent and in addition enabled. We require that the following two axioms hold:

T4 *(Urgency)* For every $\tau \in \mathcal{T}$, $t \in dom(\tau)$ and $a \in A$: if a is urgent in $\tau(t)$ then $t = \tau.ltime$.

T5 *(Maximality)* For every $\tau \in \mathcal{T}$, if τ is maximal and finite then τ is right-closed and some $a \in A$ is urgent in $\tau.lval$.

Axiom **T4** states that as soon as an action becomes urgent, this action or some other action that is enabled has to occur immediately[1]. Axiom **T5** states that each maximal and finite trajectory enables an urgent action at the end.

[1] The reader may wonder why we do not impose a stronger axiom stating that if an action becomes urgent this action or some other *urgent* action has to occur immediately. Such an approach, which involves the use of

Timed automata with urgency can be conveniently specified in a slight variation of the TIOA language [15].

Example 1 *To illustrate this language, we consider the simple model of a train displayed in Figure 1. The automaton runs cyclically through states $start$, $light$ and $gate$. After spending between $(2, 5]$ time units in $start$ the automaton jumps to $light$, then within $(5, 10]$ time units after arrival in $light$ the automaton jumps to $gate$, and after exactly 2 time units in $gate$ the automaton returns to the initial state $start$.*

```
type $controlType$ = enumeration of $start, light, gate$
automaton $Train$
  states        $control : controlType$ initially $start$
                clock $x$ initially $0$
  signature     external $coming, approaching, passing$
  transitions   external $coming$
                  pre $x > 2 \wedge control = start$
                  urgent when $x \geq 5$
                  eff $control := light; x := 0$
                external $approaching$
                  pre $x > 5 \wedge control = light$
                  urgent when $x \geq 10$
                  eff $control := gate; x := 0$
                external $passing$
                  pre $x = 2 \wedge control = gate$
                  urgent when $true$
                  eff $control := start; x := 0$
```

Figure 1. A simple model of a train.

The definitions of the signature, state variables, initial states, and transition in our language are similar to their counterparts in the IOA language. We refer to the IOA user guide and reference manual [9] for additional information on this part of the language.[2] In this article, we consider only two types of state variables, which differ in their dynamic types: *discrete* variables (such as $control$), whose value remains unchanged along a trajectory, and *clocks* (such as x), which are real-valued variables whose value increases with rate 1 along a trajectory.[3] The set of states consists of all valuations of the state variables $\vec{v}$. At the syntactic level, we have a finite number of action names b and each action name comes with a list $\vec{h}$ of formal parameters. At the semantic level, the set of actions consists of pairs of an action name b and a valuation $\mathbf{h}$ of the parameters $\vec{h}$. The transition relation is defined via a finite number of transition definitions. Each transition definition consists of an action name b, a list $\vec{h}$ of formal parameters, a **pre**condition predicate $pre(\vec{v}, \vec{h})$ that defines from which states an action $b(\vec{h})$ is enabled, an **urgent when** predicate $urg(\vec{v}, \vec{h})$ that specifies when that action becomes urgent, and an **eff**ect predicate $\mathit{eff}(\vec{v}, \vec{h}, \vec{v}')$ specifying to which states $\vec{v}'$ one may jump after doing action $b(\vec{h})$ in state $\vec{v}$. If no parameters are mentioned then the parameter list is assumed to be empty, if no precondition is mentioned then it is implicitly assumed to equal true, and if no urgency predicate is mentioned this is assumed to equal false. We further assume that the effect relation is total, in the sense that for each state $\mathbf{x}$ and parameter valuation $\mathbf{h}$ such that $pre(\mathbf{x}, \mathbf{h})$ holds, there exists at least one state $\mathbf{x}'$ such that $\mathit{eff}(\mathbf{x}, \mathbf{h}, \mathbf{x}')$ holds. If the effect predicate is defined using (deterministic) assignments, such as in Figure 1, this property trivially holds. The set of trajectories is defined implicitly. For $\mathbf{x}$ a state and t a non-negative real number, let $\mathbf{x} \oplus t$ be the state given by

$$\mathbf{x} \oplus t(v) \quad \triangleq \quad \begin{cases} \mathbf{x}(v) & \text{if } v \text{ is discrete} \\ \mathbf{x}(v) + t & \text{if } v \text{ is a clock.} \end{cases}$$

The state $\mathbf{x} \ominus t$ is defined similarly: replace $+$ by $-$ in the definition of $\oplus$. For $\mathbf{x}$ a state and I a time interval that starts with 0, a *pretrajectory* from $\mathbf{x}$ over I is a function $\tau : I \rightarrow Q$ such that for each $t \in I$, $\tau(t) = \mathbf{x} \oplus t$. The set of trajectories is defined to be the set of all pretrajectories τ satisfying that if some action a is urgent in some state $\tau(t)$, for t in the domain of τ, $t = \tau.ltime$.

Example 2 *Figure 2 gives another example of a specification in our language. It is a model of a reliable FIFO channel that delivers its messages within a certain time bound, represented by the automaton parameter b, which is a positive real number.*

The other automaton parameter M represents the type of messages communicated by the channel. The states of the automaton are valuations of the state variables $queue$ and now. The discrete variable $queue$ holds a finite sequence of pairs consisting of a message that has been sent and its delivery deadline. The clock variable now records the current real time. A $send(m)$ action, which is always enabled and never becomes urgent, adds to the queue a new pair whose first component is m and whose second component is the deadline $now + b$. A $receive(m)$ action can occur when m is the first message in the queue and it results in the removal of the first message from the queue. The $receive(m)$ action becomes urgent when the delivery deadline u of the first message equals the current time now.

priorities, has been studied in [12]. It is well-known that priorities are incompatible with a trace based semantics. We feel that, for all practical purposes, axiom **T4** allows us to specify the desired urgency properties, while it is still fully compatible with the trace based semantics which has been the preferred semantic model for (timed) I/O automata since the first paper from 1987 [20].

[2] Since the emphasis in this article is on urgency we decided not to present all datatype definitions. At this point, our specifications are (deliberately) a bit sloppy.

[3] Our results easily generalize to more general dynamic types and continuous behavior defined by arbitrary differential equations and inclusions, such as studied e.g. in [16, 19].

```
automaton $Channel(b, M)$ where $b \in \mathsf{R}^+$
  states       $queue \in (M \times \mathsf{R})^*$ initially empty
               clock $now$ initially $0$
  signature    external $send(m), receive(m)$ where $m \in M$
  transitions  external $send(m)$
                   eff add $(m, now + b)$ to the end of $queue$
               external $receive(m)$
                   pre $\exists u : (m, u)$ is first element of $queue$
                   urgent when $\exists u : (m, u) \in queue$ and $now \geq u$
                   eff remove first element of $queue$
```

Figure 2. Time-bounded channel.

By construction the set of trajectories denoted by a specification in our language satisfies axioms **T0**-**T4**. However, the example below shows that axiom **T5** does not need to hold in general.

Example 3 *The timed automaton specified in Figure 3 has a transition with precondition $x > 4 \wedge b = \mathsf{false}$ and urgency predicate* true. *Axiom* **T5** *does not hold, since time can only advance up to $x = 4$ but at that time the transition is not (yet) enabled.*

```
automaton $\mathcal{A}$
  states       $b : Bool$ initially false
               clock $x$ initially $0$
  signature    external $a$
  transitions  external $a$
                   pre $x > 4 \wedge b = \mathsf{false}$
                   urgent when true
                   eff $b := \mathsf{true}$
```

Figure 3. A counterexample to axiom T5.

In order to avoid the counterexample of Figure 3, it is sufficient that certain predicates derived from the transition definitions are left-closed in the sense of [3]. For each transition definition tr

```
$b(\vec{h})$     pre $pre(\vec{v}, \vec{h})$
                 urgent when $urg(\vec{v}, \vec{h})$
                 eff $eff(\vec{v}, \vec{h}, \vec{v}')$
```

let predicate $Urg(tr)$ be given by

$$Urg(tr)(\vec{v}, \vec{h}) \quad \triangleq \quad \exists \vec{h} : pre(\vec{v}, \vec{h}) \wedge urg(\vec{v}, \vec{h}) \qquad (2)$$

Following [3], we define a state predicate φ to be *left-closed* if, for all $\vec{v}$,

$$\neg\varphi(\vec{v}) \implies \exists \epsilon > 0 \, \forall \epsilon' \leq \epsilon : \neg\varphi(\vec{v} \oplus \epsilon') \qquad (3)$$

In practice, left-closedness can be easily obtained by only using non-strict lower bounds on clocks. For instance, $x \geq 4 \wedge b = \mathsf{false}$ is left-closed but $x > 4 \wedge b = \mathsf{false}$ is not. We can now formally state the following theorem.

Theorem 1 *If the predicate $\bigvee_{tr} Urg(tr)$ is left-closed then axiom* **T5** *holds.*

Proof: Suppose that τ is a maximal and finite trajectory. Assume that the domain of τ is right-open. Then, by **T4**, nowhere on τ an action becomes urgent. But this means that the extension of τ with a single state at the end gives a legal trajectory, thus contradicting the assumption that τ is maximal. Hence, without loss of generality, we may assume that the domain of τ is right-closed. Assume that no action a is urgent in $\tau.lval$. This means that the disjunction $\bigvee_{tr} Urg(tr)$ does not hold in $\mathbf{x}$. But this means that there exists a small ϵ-extension of τ in which no action a is urgent. This extension is then a legal trajectory, which contradicts with the (assumed) maximality of τ. ■

2.4. Adding an I/O Distinction

In this section, we further refine the model of timed automata by distinguishing between input and output actions as in [16].

A *timed I/O automaton with urgency* is a quadruple $(\mathcal{A}, U, I, O)$ where $(\mathcal{A}, U)$ is a timed automaton with urgency, with $\mathcal{A} = (X, Q, \Theta, E, H, \mathcal{D}, \mathcal{T})$. I and O partition E into input and output actions, that is $E = I \cup O$ and $I \cap O = \emptyset$. Actions in $H \cup O$ are called locally controlled. We write $L \triangleq H \cup O$ and $A \triangleq E \cup H$. We require that the following axiom holds:

E0 *(Inputs not urgent)* For every $\mathbf{x} \in Q$ and every $a \in I$, $U(\mathbf{x}, a) = \mathsf{false}$.

E1 *(Input action enabling)* For every $\mathbf{x} \in Q$ and every $a \in I$, there exists $\mathbf{x}' \in Q$ such that $(\mathbf{x}, a, \mathbf{x}') \in \mathcal{D}$.

The input actions are assumed not to be under the automaton's control—they just arrive from the outside—while the automaton itself specifies what output and internal actions should be performed. In line with these intuitions, axiom **E0** states that input actions never become urgent. Axiom **E1** is the usual input enabling condition of ordinary I/O automata [20]; it says that a TIOA with urgency is able to accomodate an input action whenever it arrives. At the syntactic level, a sufficient condition for axioms **E0** and **E1** to hold is that, in each transition definition for an input action, the precondition is true and the urgency predicate is false.

A desirable property for models of real-time systems is *time reactivity*. This means that in each state, either time is allowed to advance forever, or time may advance for a while up to a point where the system is prepared to react with some locally controlled action. In [16], an axiom **E2** is required for timed I/O automata which captures this property:

E2 *(Time-passage enabling)* For every $\mathbf{x} \in Q$, there exists $\tau \in \mathcal{T}$ such that $\tau.fval = \mathbf{x}$ and either (1) $\tau.ltime = \infty$, or (2) τ is right-closed and some locally controlled action $l \in L$ is enabled in $\tau.lval$.

For a TIOA with urgency, time reactivity is implied by the other axioms.

Theorem 2 *Each timed I/O automaton with urgency satisfies axiom* **E2**.

2.5. Composition

We say that timed automata $\mathcal{A}_1$ and $\mathcal{A}_2$ are *compatible* if they have no state variables in common, and if neither automaton has an internal action that is an action of the other automaton. If $\mathcal{A}_1$ and $\mathcal{A}_2$ are compatible then their *composition* $\mathcal{A}_1||\mathcal{A}_2$ is defined formally to be the timed automaton $\mathcal{A} = (X, Q, \Theta, E, H, \mathcal{D}, \mathcal{T})$ where

- $X = X_1 \cup X_2$.
- $Q = \{\mathbf{x} \in val(X) | \mathbf{x}\lceil X_i \in Q_i, i \in \{1,2\}\}$.
- $\Theta = \{\mathbf{x} \in Q | \mathbf{x}\lceil X_i \in \Theta_i, i \in \{1,2\}\}$.
- $E = E_1 \cup E_2$ and $H = H_1 \cup H_2$.
- For each $\mathbf{x}, \mathbf{x}' \in Q$ and each $a \in A$, $\mathbf{x} \xrightarrow{a}_{\mathcal{A}} \mathbf{x}'$ iff for $i \in \{1,2\}$, either (1) $a \in A_i$ and $\mathbf{x}\lceil X_i \xrightarrow{a}_i \mathbf{x}'\lceil X_i$, or (2) $a \notin A_i$ and $\mathbf{x}\lceil X_i = \mathbf{x}'\lceil X_i$.
- $\tau \in \mathcal{T} \Leftrightarrow \tau \downarrow X_i \in \mathcal{T}_i, i \in \{1,2\}$.

We refer to [17] for a proof that $\mathcal{A}_1||\mathcal{A}_2$ is a timed automaton, that is, the above structure satisfies axioms **T0**-**T3**.

Two timed automata with urgency, $(\mathcal{A}_1, U_1)$ and $(\mathcal{A}_2, U_2)$, are *compatible* if the underlying timed automata $\mathcal{A}_1$ and $\mathcal{A}_2$ are compatible. In this case, the *composition* is defined to be the structure $(\mathcal{A}, U)$, where $\mathcal{A} = \mathcal{A}_1||\mathcal{A}_2$ and U is given by

$$U((\mathbf{x}_1, \mathbf{x}_2), a) \quad = \quad U_1(\mathbf{x}_1, a) \vee U_2(\mathbf{x}_2, a),$$

where by convention $U_i(\mathbf{x}_i, a) = \mathsf{false}$ if a is not in the signature of $\mathcal{A}_i$. So an action is urgent in a state of the composed system iff it is urgent in one of the component states. In general, the composition is not a timed automaton with urgency. The problem is due to axiom **T5**: if, for instance, we compose a system in which action a becomes urgent at time 1 with a system that has a in its signature but without any a-transition, then the composed system has a maximal trajectory of length 1 in which no transition is enabled. Several papers address the issue of how timelock freedom (or more generally, liveness) can be preserved by composition, see for instance [5, 4, 3]. In this article, we present one simple but useful result along these lines: the class of timed I/O automata with urgency *is* closed under composition.

We say that two timed I/O automata with urgency, $(\mathcal{A}_1, U_1, I_1, O_1)$ and $(\mathcal{A}_2, U_2, I_2, O_2)$, are *compatible* if the underlying timed automata $\mathcal{A}_1$ and $\mathcal{A}_2$ are compatible, and also they have no output actions in common. A consequence of these conditions is that each action is controlled by at most one component. In this case, the *composition* is defined to be the structure $(\mathcal{A}, U, I, O)$, where $(\mathcal{A}, U)$ is the composition of $(\mathcal{A}_1, U_1)$ and $(\mathcal{A}_2, U_2)$, $I = (I_1 \cup I_2) - (O_1 \cup O_2)$, and $O = O_1 \cup O_2$. That is, an external action of the composition is classified as an output if it is an output of one of the component automata, otherwise it is classified as an input.

Theorem 3 *The composition of two compatible timed I/O automata with urgency is again a timed I/O automaton with urgency.*

3. Expressivity

In this section, we compare the expressivity of urgency predicates with that of the deadline predicates of [24, 4], the stopping conditions of [16], and the invariants as used e.g. in [14, 2, 18].

3.1. Deadline Predicates

Instead of using urgency predicates, we could follow the approach of Sifakis et al [24, 4] even more closely by using deadline predicates. This would mean that, for a given action name b with parameters $\vec{h}$, besides the precondition $pre(\vec{v}, \vec{h})$ and the effect $eff(\vec{v}, \vec{v}', \vec{h})$, also a *deadline* predicate $dl(\vec{v}, \vec{h})$ is specified such that $dl(\vec{v}, \vec{h}) \implies pre(\vec{v}, \vec{h})$ holds. The semantics of a deadline predicate is that if, for some $\vec{h}$, the state predicate

$$dl(\vec{v}, \vec{h}) \qquad (4)$$

becomes true at a time point t in a trajectory, then t must be the limit time of that trajectory.

Clearly, any definition of a timed automaton with urgency predicates can be transformed into an equivalent (in the sense that the defined automata are semantically equal) definition with deadline predicates by replacing each urgency predicate $urg(\vec{v}, \vec{h})$ by a deadline predicate $pre(\vec{v}, \vec{h}) \wedge urg(\vec{v}, \vec{h})$. Conversely, any definition with deadline predicates can be transformed into an equivalent definition with urgency predicates by replacing each deadline predicate $dl(\vec{v}, \vec{h})$ by an identical urgency predicate $dl(\vec{v}, \vec{h})$. Studying the examples in Figure 1 and Figure 2, and the examples in [17] indicates that the use of urgency predicates leads to slightly shorter specifications than the use of deadlines.

3.2. Stopping Conditions

Another alternative for urgency predicates are the stopping conditions as used in [16]. A stopping condition is a state predicate $sc(\vec{v})$ such that if $sc(\vec{v})$ becomes true at a time point t in a trajectory, then t must be the limit time of that trajectory.

We again checked the examples from Figure 1, Figure 2 and [17], and in each case urgency predicates lead to shorter and (in our view) more natural specifications than stopping conditions. Figure 4, for instance, shows how the transitions and trajectories of the example of Figure 1 can be rewritten using a stopping condition. The disadvantages should be clear: upper bounds are no longer specified next to the corresponding lower bounds, and parts of the preconditions have to be repeated in the stopping condition.

```
transitions    external $coming$
                  pre $x > 2 \wedge control = start$
                  eff $control := light; x := 0$
               external $approaching$
                  pre $x > 5 \wedge control = light$
                  eff $control := gate; x := 0$
               external $passing$
                  pre $x = 2 \wedge control = gate$
                  eff $control := start; x := 0$
trajectories   stops when
                  $(control = start \wedge x \geq 5) \vee$
                  $(control = light \wedge x \geq 10) \vee$
                  $(control = gate \wedge x = 2)$
```

Figure 4. The train model defined using a stopping condition.

Any definition of a timed automaton with urgency predicates can be transformed into an equivalent definition with stopping conditions by replacing the urgency predicates by a single stopping condition that is the disjunction of the formula $pre(\vec{v}, \vec{h}) \wedge urg(\vec{v}, \vec{h})$, for all transition definitions.

Stopping conditions are more expressive than urgency predicates since they allow one to define timed automata that are not time reactive and in which "the universe" may come to a halt. Figure 5 gives an example. Of course this is a form of additional expressivity that we would rather not have! For a timed automaton definition with a stopping condition $sc(\vec{v})$ it seems reasonable to require that the following variation of axiom **T5** holds:

```
automaton $Doomsday$
  states         clock $x$ initially $0$
  trajectories   stops when    $x = 1$
```

Figure 5. A time deadlock.

T5' *(Maximality)* For every $\tau \in \mathcal{T}$, if τ is maximal and finite then τ is right-closed, $sc(\tau.lval)$ and some (locally controlled) $a \in A$ is enabled in $\tau.lval$.

If this property holds, the specification can be transformed into an equivalent specification with urgency predicates: in case there is no I/O distinction we just add an urgency predicate $sc(\vec{v})$ to each transition definition, if there is an I/O distinction we add urgency predicate $sc(\vec{v})$ to each locally controlled transition and urgency predicate false to each input transition.

3.3. Invariants

A popular way to specify progress properties, which has been advocated in [14, 2] and implemented in UPPAAL [18], is the use of invariants. An *invariant* is a state predicate $inv(\vec{v})$ that is required to hold for all states along all trajectories. The transitions and trajectories of a timed automaton with invariants looks exactly like the automaton with stopping condition as shown in Figure 4. Again, like stopping conditions, invariants allow one to define timed automata that are not time reactive, a clear disadvantage of these specification styles. The example of Figure 5, for instance, can easily be encoded using invariants (replace the stopping condition by an invariant $x \leq 1$).

Invariants also allow one to specify strict upper bounds on the timing of events, as illustrated in Figure 6. The same timed automaton can not be specified using urgency predicates, for the simple reason that it has a maximal trajectory that is right-open, which is in violation with axiom **T5**. If we are willing to consider timed automata up to some suitable equivalence (for instance, the trace equivalence defined in [16]) then it is possible to specify strict upper bounds with urgency predicates, but this requires the use of auxiliary variables and unbounded nondeterminism. Figure 7 illustrates the specification of a strict upper bound with an urgency predicate. The idea is to choose nondeterministically a value in the interval $[0, 1)$ and then make a urgent when time has reached this value. Apart from the fact that the second specification is less intuitive, the use of unbounded nondeterminism will constitute a serious obstacle for automatic verification methods. In all practical applications of timed automata that we are aware of, the use of only non

```
automaton $BeforeOne$
  states         discrete $b : Bool$ initially false
                 clock $x$ initially $0$
  signature      external $a$
  transitions    external $a$
                    pre $b = \text{false}$
                    eff $b := \text{true}$
  trajectories   invariant $x < 1 \vee b = \text{true}$
```

Figure 6. Specification of a strict upper bound on timing with an invariant.

```
automaton $BeforeOne'$
  states        $b : Bool$ initially false
                $t : \mathsf{R}$ initially $0 \leq t < 1$
                clock $x$ initially $0$
  signature     external $a$
  transitions   external $a$
                  pre $b = \mathsf{false}$
                  urgent when $x = t$
                  eff $b := \mathsf{true}$
```

Figure 7. Specification of a strict upper bound on timing with urgency.

strict upper bounds on timing is not a restriction. For applications where use of strict upper bounds is essential, use of invariants is probably more appropriate than use of urgency predicates.

Timed automata with urgency predicates can (in many cases) be translated to equivalent timed automata with invariants. Robson [23] describes how a fragment of TIOA with urgency predicates can be translated to the input language of UPPAAL.[4] Below we discuss a more general translation scheme. We say that a state predicate $\varphi(\vec{v})$ is *stable (under time progress)* if $\varphi(\vec{v}) \implies \forall d > 0 : \varphi(\vec{v} \oplus d)$.

Typically, a predicate will be stable if it only involves lower bounds on clocks and no upper bounds. The *lower hull* of state predicate $\varphi(\vec{v})$ is the set of valuations given by

$$LH(\varphi) \triangleq \{\mathbf{x} \mid \varphi(\mathbf{x}) \wedge \exists \epsilon > 0\, \forall 0 < \epsilon' \leq \epsilon : \neg\varphi(\mathbf{x} \ominus \epsilon')\}$$

The upper hull of a state predicate can be defined similarly, just replace $\ominus$ by $\oplus$ in the above definition. If φ only involves (non-strict) lower bounds on clocks then the lower hull can easily be expressed again as a predicate by replacing the $\geq$ signs with $=$.

Now consider a definition of a timed automaton with urgency predicates such that all predicates $Urg(tr)$ are left-closed and stable. An equivalent timed automaton with invariants can be obtained by replacing the urgency predicates with the invariant

$$inv \quad = \quad \neg\left(\bigvee_{tr} Urg(tr)\right) \vee LH\left(\bigvee_{tr} Urg(tr)\right),$$

provided that the state predicate inv holds initially and after each discrete transition, i.e.,

$$pre(\vec{v}, \vec{h}) \wedge \mathit{eff}(\vec{v}, \vec{h}, \vec{v}') \implies inv(\vec{v}').$$

The proof of the equivalence is straightforward and left to the reader.

Any timed automaton definition with right-closed invariants can be easily translated to a timed automaton with stopping conditions: the stopping condition is defined to be (a predicate denoting) the upper hull of the invariant. The translation scheme of Section 3.2 can then be used (provided axiom **T5'** holds) to translate the resulting timed automaton with stopping conditions to a timed automaton with urgency.

4. Proving Invariant Properties

In this section, we discuss how to establish invariant properties for specifications that involve urgency predicates. It is important to distinguish invariant properties from the invariant assertions that were discussed in the previous section as a construct to specify progress. An invariant property is a state predicate that *holds* for all reachable states of a given system. An invariant in the sense of previous section is an assertion that is actually used to *define* (the trajectories of) a system. Any invariant in the sense of the previous section is actually an invariant property of the system that it helps to define. The converse implication typically does not hold.

An *execution fragment* of a timed automaton $\mathcal{A}$ is a sequence $\alpha = \tau_0\ a_1\ \tau_1\ a_2\ \tau_2 \ldots$, where each a_i is an action of $\mathcal{A}$, each τ_i is a trajectory of $\mathcal{A}$, and for every i, $\tau_i.lval \overset{a_{i+1}}{\to} \tau_{i+1}.fval$. An execution fragment records what happens during a particular run of a system, including all the discrete state changes and all the changes that occur while time advances. An *execution* is an execution fragment whose first state is a start state of $\mathcal{A}$. A state is *reachable* in $\mathcal{A}$ if it is the last state of the last trajectory of a finite execution of $\mathcal{A}$. A state predicate φ is an *invariant* of $\mathcal{A}$ if it holds for all reachable states of $\mathcal{A}$.

In order to prove that an assertion φ is an invariant of $\mathcal{A}$, it suffices to prove that it holds initially and is preserved by all discrete transitions as well as by all $time(d)$ steps defined by

$$\begin{aligned} \mathbf{x} \xrightarrow{time(d)} \mathbf{x}' \quad \triangleq \quad & \exists \tau \in \mathcal{T} : \tau.fval = \mathbf{x}\ \wedge \\ & \tau.ltime = d \wedge \tau.lval = \mathbf{x}'. \end{aligned}$$

If we manage to give a simple and tractable characterization of the $time(d)$ predicate, then all the invariant proof techniques which are presented (for instance) in [21] become available in our setting.

Let tr be a transition definition for an action name b with parameter $\vec{h}$ with precondition $pre(\vec{v}, \vec{h})$, urgency predicate $urg(\vec{v}, \vec{h})$, and effect predicate $\mathit{eff}(\vec{v}, \vec{h}, \vec{v}')$. For $d \geq 0$, the *time progress* predicate $tp(\vec{v}, tr, d)$ expresses that transition tr permits time to advance with an amount d from state $\vec{v}$.

[4] The UPPAAL syntax for invariant predicates is rather restricted. For each individual location the invariant is a conjunction of conditions of the form $x \leq e$ or $x < e$ where x is a clock and e is an expression that evaluates to an integer. This restriction forces Robson to split locations as part of her translation.

The predicate is formally defined in terms of the $Urg(tr)$ predicate of (2):

$$\begin{aligned} tp(\vec{v}, tr, d) &\triangleq \forall 0 \leq e < d : \neg Urg(tr)(\vec{v} \oplus e, \vec{h}) \quad (5) \\ &\triangleq \forall 0 \leq e < d,\ \forall h : \\ &\quad pre(\vec{v} \oplus e, \vec{h}) \implies \neg urg(\vec{v} \oplus e, \vec{h}) \end{aligned}$$

Using the time progress predicates, we characterize the time advance steps $time(d)$ as follows:

$$\begin{array}{l} time(d) \\ \quad \textbf{pre}\ \bigwedge_{tr} tp(\vec{v}, tr, d) \\ \quad \textbf{eff}\ \vec{v} := \vec{v} \oplus d \end{array}$$

In many cases it is possible to simplify the time progress predicates, by eliminating the universal quantifications from their definition. As an example, consider the timed automaton of Figure 1. The time progress predicate for the $coming$ transition is

$$\begin{array}{ll} \multicolumn{2}{l}{\forall 0 \leq e < d : \neg(x + e > 2 \wedge control = start \wedge x + e \geq 5)} \\ \iff & \forall 0 \leq e < d : \neg(control = start \wedge x + e \geq 5) \\ \iff & \neg(control = start \wedge x + d > 5) \\ \iff & control = start \implies x + d \leq 5 \end{array}$$

Similarly, the time progress predicates for the $approaching$ and $passing$ transitions can be written resp. as $control = light \implies x + d \leq 10$ and $control = gate \implies x + d \leq 2$ respectively. With these characterizations it is trivial to prove, for example, that $control = start \implies x \leq 5$ is inductive (and hence an invariant): it holds initially, and it is preserved by all discrete transitions and all $time(d)$ steps.

The above quantifier elimination can be generalized under some reasonable assumptions. If $\varphi(\vec{v})$ is a state predicate then we write $post(\varphi)(\vec{v})$ for the state predicate that holds for states that have a time predecessor satisfying φ:

$$post(\varphi)(\vec{v}) \triangleq \exists \vec{w}\ \exists e > 0 : (\vec{v} = \vec{w} \oplus e) \wedge \varphi(\vec{w}) \quad (6)$$

If φ only involves lower bounds on clocks, then $post(\varphi)$ can typically be obtained from φ by making these lower bounds strict, so quantifier elimination from $post(\varphi)$ is easy. Hence, if preconditions and urgency predicates only involve lower bounds on clocks (which appears to be a good specification style anyway), then their conjunction is stable one may use the following lemma to eliminate the quantification over e from the time progress predicate (5).

Lemma 2 *Let φ be a state predicate that is stable under time progress. Then*

$$(\forall 0 \leq e < d : \neg\varphi(\vec{v} \oplus e)) \Leftrightarrow \neg post(\varphi)(\vec{v} \oplus d) \qquad (7)$$

Proof: Equivalence (7) can be rewritten into $(\exists 0 \leq e < d : \varphi(\vec{v} \oplus e)) \Leftrightarrow post(\varphi)(\vec{v} \oplus d)$ We prove both implications:

- $\Rightarrow$ Assume $\varphi(\vec{v} \oplus e)$, for certain $e \in [0, d)$. Then $post(\varphi)(\vec{v} \oplus d)$ holds since there is a state, namely $\vec{v} \oplus e$, that is a time predecessor of $\vec{v} \oplus d$ and in which φ holds.
- $\Leftarrow$ Assume $post(\varphi)(\vec{v} \oplus d)$. Then by (6) there exists an $e' > 0$ and a state $\vec{w}$ such that $\varphi(\vec{w})$ holds and $\vec{v} \oplus d = \vec{w} \oplus e'$. Depending on the relationship between d and e' we have two cases:

  Case $e' \leq d$**:** Then $\vec{v} \oplus (d - e') = \vec{w}$. Choose $e = d - e'$. Then $d \in [0, d)$ and $\varphi(\vec{v} \oplus e)$.

  Case $d < e'$**:** Then $\vec{v} = \vec{w} \oplus (e' - d)$. Since $\varphi(\vec{w})$ holds and φ is stable under time progress, also $\varphi(\vec{v})$ holds. So we may choose $e = 0$ to obtain $\varphi(\vec{v} \oplus e)$, as required. ■

5. Concluding Remarks

In this article, we introduced a notion of urgency predicates and compared it with three other constructs for specifying progress properties that have been proposed in the literature: invariants, stopping conditons and deadlines. We showed that under some rather realistic assumptions (use of clock variables, no strict upper bounds on progress, absence of time deadlocks,...) the four notions are equally expressive. Nevertheless, a clear advantage of deadlines and urgency predicates in practice is that one gets absence of time deadlocks (time reactivity) for free. A potential advantage of invariants is that they allow one to bound the time at which a (locally controlled) action occurs by a right-open interval. However, we are not aware of practical applications in which this feature is really needed. We argued that if one uses a precondition/effect style specification language, urgency predicates lead to shorter and more natural specifications than any of the other constructs, in particular invariant. In the graphical syntax used by e.g. UPPAAL, the use of urgency/deadline predicates would not lead to shorter specifications than the use of invariants. Typically, in first case one will decorate an edge of the graph (i.e., a transition) with a label $x \geq 4$, and in the second case a label $x \leq 4$ will be attached to a vertex of the graph (i.e., a location). But whereas the use of invariants may easily lead to time deadlocks, urgency/deadline predicates only stop time if there is a good reason for it, that is a specific transition that must be taken, and in this manner time deadlocks are avoided.

Folklore has it that urgency/deadline predicates are more difficult to implement in model checkers than invariants because they easily lead to non-convex zones. Non-convex zones indeed arise in the implementation of timed automata with deadlines in the IF toolset [6]. In particular, time transitions may lead from one convex zone to several convex zones (not only one, as in standard timed automata with invariants). When such a situation arises in IF, the non-convex zone is automatically split in several, possibly overlapping, convex zones. The main reason why non-convex zones do not arise in standard timed automata with invariants (such as those implemented in UPPAAL) is that rather strong restrictions are imposed on the syntax of invariants. Only conjunctions of upper bounds on clocks—where the bounds are given by integer expressions—are allowed. If similar restrictions would be imposed in a syntax for urgency predicates, then no non-convex zones would arise in that setting either! More specifically, one would have to require that each urgency predicate is the disjunction of lower bounds on clocks, where the (non-strict) bounds are given by integer expressions. In addition, the urgency predicate of an input action $a?$ should always be false. We think it would be a clear improvement to the current version of UPPAAL (3.4.7) to add such a restricted notion of urgency predicates to the syntax, replacing the notion of an urgent channel. Adding general urgency predicates to UPPAAL would of course also be a possibility, but this would require splitting of zones as in the IF toolset.

In the setting that we studied, urgency predicates appear to be a very nice way to specify progress properties, with clear advantages over some other constructs that have been advocated in the literature. Some remaining questions for future research are: (1) Exploration of proof rules to reason with urgency predicates in simulations and liveness proofs. (2) Establish versions of the compositionality results of [5, 4, 3] in the setting of this paper. (3) Extension of our specification language and expressiveness results to a hybrid setting in which besides clocks also other continuously evolving variables are allowed.

Acknowledgements Thanks to Nancy Lynch and Dilsun Kaynar for detailled comments on an earlier version of this note, and to Marius Bozga for answering some of our questions about IF.

References

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

[2] R. Alur and T. Henzinger. Real-time system = discrete system + clock variables. *First AMAST Workshop on Real-Time System Development*, pages 1–29. World Scientific, 1994.

[3] S. Bornot, G. Gößler, and J. Sifakis. On the construction of live timed systems. In *6th Proceeding of TACAS'00*, volume 1785 of *LNCS*, pages 172–202. Springer-Verlag, 2000.

[4] S. Bornot and J. Sifakis. An algebraic framework for urgency. *Info. and Comp.*, 163:172–202, 2000.

[5] H. Bowman. Modelling timeouts without timelocks. In *Proceeding of ARTS'99*, LNCS, page 20. Springer-Verlag, 1999.

[6] M. Bozga, S. Graf, and L. Mounier. IF-2.0: A validation environment for component-based real-time systems. In *Proceeding of CAV'02*, volume 2404 of *LNCS*, pages 343–348, Copenhagen, Denmark, 2002. Springer.

[7] P. D'Argenio and B. Gebremichael. The coarsest congruence for timed automata with deadlines contained in bisimulation. In *16th International Conference on Concurrency Theory (CONCUR05)*, August 2005. to appear.

[8] P. D'Argenio, H. Hermanns, J.-P. Katoen, and R. Klaren. MoDeST - a modelling and description language for stochastic timed systems. In *Proceeding of PAPM-PROBMIV 2001*, volume 2708 of *LNCS*, pages 87–104. Springer, 2003.

[9] S. Garland, N. Lynch, J. Tauber, and M. Vaziri. IOA user guide and reference manual, 2003.

[10] B. Gebremichael and F. Vaandrager. Specifying urgency in timed I/O automata. Technical Report NIII-R0459, ICIS. Radboud University Nijmegen, 2004.

[11] R. Gomez and H. Bowman. Discrete Timed Automata and MONA: Description, Specification and Verification of a Multimedia Stream. In *Proceeding of FORTE'03*, volume 2767 of *LNCS*, pages 177–192. Springer, 2003.

[12] G. Gössler and J. Sifakis. Composition for component-based modeling. *Sci. Comput. Program.*, 55(1-3):161–183, 2005.

[13] S. Graf and I. Ober. A real-time profile for UML and how to adapt it to SDL. In *Proceeding of SDL Forum'03*, volume 2165 of *LNCS*, pages 55–76. Springer, 2001.

[14] T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Info. and Comp.*, 111:193–244, 1994.

[15] D. Kaynar, N. Lynch, and S. Mitra. Specifying and proving timing properties with TIOA tools. In *25th Proceeding of RTSS'04 WIP*. IEEE Computer Society, 2004.

[16] D. Kaynar, N. Lynch, R. Segala, and F. Vaandrager. A framework for modelling timed systems with restricted hybrid automata. In *24th Proceedings of RTSS'03*, pages 166–178. IEEE Computer Society Press, 2003.

[17] D. Kaynar, N. Lynch, R. Segala, and F. Vaandrager. The theory of timed I/O automata. Technical Report MIT-LCS-TR-917, MIT Laboratory for Computer Science, Cambridge, MA, 2003.

[18] K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1(1–2):134–152, Oct. 1997.

[19] N. Lynch, R. Segala, and F. Vaandrager. Hybrid I/O automata. *Info. and Comp.*, 185(1):105–157, 2003.

[20] N. A. Lynch and M. R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *6th Proc. of the Principles of Distributed Computing*, pages 137–151. MIT, 1987.

[21] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer, 1995.

[22] M. Merritt, F. Modugno, and M. Tuttle. Time constrained automata. In *6th Proceeding of CONCUR'91*, volume 527 of *LNCS*, pages 408–423. Springer-Verlag, 1991.

[23] C. Robson. TIOA and UPPAAL. Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, 2004.

[24] J. Sifakis and S. Yovine. Compositional specification of timed systems (extended abstract). In *Proceeding of STACS*, volume 1046 of *LNCS*, pages 347–359. Springer, 1996.