A Proof System for Timed Automata

Huimin Lin[1]* Wang Yi[2]

[1]Laboratory for Computer Science
Institute of Software, Chinese Academy of Sciences
Email: lhm@ox.ios.ac.cn

[2]Department of Computer Systems
Uppsala University
Email: yi@docs.uu.se

Abstract. A proof system for timed automata is presented, based on a CCS-style language for describing timed automata. It consists of the standard monoid laws for bisimulation and a set of inference rules. The judgments of the proof system are *conditional equations* of the form $\phi \rhd t = u$ where ϕ is a clock constraint and t, u are terms denoting timed automata. It is proved that the proof system is complete for timed bisimulation over the recursion-free subset of the language. The completeness proof relies on the notion of *symbolic timed bisimulation*. The axiomatisation is also extended to handle an important variation of timed automata where each node is associated with an invariant constraint.

1 Introduction

Timed automata [AD94] has been recognised as a fundamental model for real time systems. By now the theory of timed automata has been well developed, but there is still one aspect missing: axiomatisation.

Timed automata extend traditional finite automata with a finite set of real-valued *clock variables* (or *clocks* for short) by annotating each transition with, in addition to an action label, a clock constraint (enabling condition) and a subset of clocks (reset set). Intuitively a timed automaton may stay at a node with clocks increasing uniformly (to model time passage), or choose a transition whose clock constraint is satisfied, make the move, reset the subset of clocks associated with the transition to zero, and arrive at the target node of the transition (to model control state switch). The explicit presence of clock variables and resetting, features that mainly associated with the so-called "imperative languages", distinguishes timed automata from process calculi such as CCS, CSP and their timed extensions which are "applicative" in nature and therefore more amenable to axiomatisation.

The aim of this paper is to propose a proof system for timed automata. We adapt the *symbolic bisimulation* technique originally developed for value-passing

* Supported by research grants from National Science Foundation of China and Chinese Academy of Sciences

processes [HL95,HL96] to the timed setting. We first present a simple CCS-style language in which each term represents a timed automaton. The language has a conditional construct $\phi{\rightarrow}t$ (read "if ϕ then t") where ϕ is a clock constraint. Action prefixing is of the form $a(\mathbf{x}).t$, meaning to perform action a and reset the clocks in $\mathbf{x}$ to zero, then behave like t. An inference system is then formulated with judgments being *conditional equations* of the form

$$\phi \rhd t = u$$

Intuitively it means that "t and u are timed bisimilar for each clock valuation satisfying ϕ". A typical inference rule takes the form:

$$\text{GUARD}\ \ \frac{\phi \wedge \psi \rhd t = u \qquad \phi \wedge \neg\psi \rhd \mathbf{0} = u}{\phi \rhd (\psi{\rightarrow}t) = u}$$

It performs a case analysis on the constraint ψ: $\psi{\rightarrow}t$ behaves like t when ψ is true, and like the inactive process $\mathbf{0}$ otherwise. Note that the guarding constraint ψ of $\psi{\rightarrow}t$ in the conclusion is *part of the object language* describing timed automata, while in the premise it is shifted to the condition part of the judgment in our *meta language* for reasoning about timed automata.

The crucial rule, as might be expected, is the one for action prefixing:

$$\text{ACTION}\ \ \frac{\phi{\downarrow_{\mathbf{xy}}}{\Uparrow} \rhd t = u}{\phi \rhd a(\mathbf{x}).t = a(\mathbf{y}).u} \qquad \mathbf{y} \cap \mathcal{C}(t) = \mathbf{x} \cap \mathcal{C}(u) = \emptyset$$

Here $\downarrow_{\mathbf{xy}}$ and $\Uparrow$ are postfixing operations on clock constraints. $\phi{\downarrow_{\mathbf{xy}}}{\Uparrow}$ is a clock constraint obtained from ϕ by first setting the clocks in $\mathbf{xy}$ to zero (operator $\downarrow_{\mathbf{xy}}$), then removing up-bounds on all clocks of ϕ (operator $\Uparrow$). Readers familiar with Hoare Logic may notice some similarity between this rule and the rule dealing with assignment there:

$$\{P[e/x]\}\ x := e\ \{P\}$$

But here the operator $\downarrow_{\mathbf{xy}}$ is slightly more complicated than substitution with zero, because clocks are required to increase uniformly. We also need $\Uparrow$ to allow time to pass indefinitely.

Traditionally axiomatisation for so-called "pure" process algebras are based on equational reasoning, i.e. "replacing equal for equal". Since timed automata involve clock constrains and clock resetting, it is not surprising that pure equational reasoning along is no longer adequate. The inference system proposed in this paper can be viewed as extending pure equational reasoning by formulating suitable rules for the specific constructs present in timed automata. It turns out that with this extension the standard monoid laws for bisimulation are sufficient for timed bisimulation, i.e. the proof system consisting of the set of inference rules and the four monoid laws are sound and complete for timed bisimulation. The proof of the completeness result relies on developing a theory of timed symbolic bisimulation which is a binary relation indexed by clock constraints. It captures the standard definition of timed bisimulation in the sense that t and

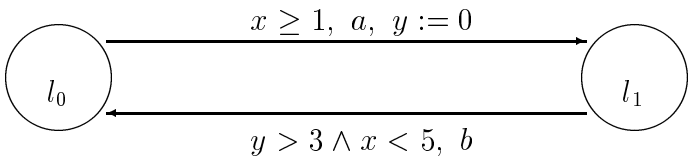

Fig. 1. A Timed Automaton.

u are symbolically bisimilar over indexing constraint ϕ if and only if they are timed bisimilar for any time valuation satisfying ϕ.

In the remaining of this section we briefly discuss related work. The language for timed automata is presented in the next section, with a symbolic operational semantics which associates each term in the language to a timed automaton. Section 3 develops a theory of symbolic bisimulation for timed automata. The proof system is presented in Section 4, together with its completeness proof. Section 5 discusses how to extend the language to include *invariants*. The paper is concluded with Section 6 where further research direction is also outlined.

Related work The first process algebra for timed automata is proposed in [WPD94] as the very first input language for the UPPAAL tool. The only previous attempt to axiomatizing timed automata we are aware is [DAB96], which develops a large set of sound axioms for timed bisimulation. However, no completeness result is reported.

On the other hand, most timed extensions of process algebras came with axiomatisation on various equivalence relations including bisimulation. Of particular interest is [Bor96] which also adapts the symbolic bisimulation technique of [HL95,HL96] to a timed process language and proposed a symbolic style proof system. As noted by the author, the language considered in that paper is quite different from timed automata as it does not involve clock variables.

2 A Language for Timed Automata

The theory of timed automata was introduced in [AD94] and has since then established as a standard model for real time systems. We first give a brief review for the readers unfamiliar with timed automata and then present an algebraic language in which each term denotes a timed automaton.

2.1 Timed Automata

A timed automaton is a standard finite-state automaton extended with a finite collection of real-valued clocks. In a timed automaton each transition is labelled with a *guard* (a constraint over clocks), a *synchronisation action*, and a *reset set* (a subset of clocks to be reset). Intuitively, a timed automaton starts an execution

with all clocks initialised to zero. Clocks increase at the same rate while the automaton stays within a node. A transition can be taken if the clocks fulfill the guard. By taking the transition, all clocks in the clock reset are set to zero, while the others are unchanged. Thus transitions occur instantaneously. Semantically, a state of an automaton is a pair of a control node and a *clock valuation*, i.e. the current setting of the clocks. Transitions in the semantic interpretation are either labelled with a synchronisation action (if it is an instantaneous switch from the current node to another) or a positive real number i.e. a time delay (if the automaton stays within a node letting time pass).

Consider the timed automaton of Figure 1. It has two control nodes l_0 and l_1 and two real–valued clocks x and y. A *state* of the automaton is of the form $(l, \langle s, t\rangle)$, where l is a control node, and s and t are non–negative reals giving the value of the two clocks x and y. Assuming that the automaton starts to operate in the state $(l_0, \langle 0, 0\rangle)$, it may stay in node l_0 for any amount of time, while the values of the clocks increase uniformly, at the same rate. Thus from the initial state, all states of the form $(l_0, \langle t, t\rangle)$ with $t \geq 0$ are reachable. However, only at the states $(l_0, \langle t, t\rangle)$, where $t \geq 1$, the edge from l_0 to l_1 is enabled. Additionally, edges are labelled with synchronization actions and simple assignments reseting clocks. For instance, when following the edge from l_0 to l_1 the action a is performed to synchronize with the environment and the clock y is reset to 0, leading to states of the form $(l_1, \langle t, 0\rangle)$, where $t \geq 1$.

For the formal definition, we assume a finite set of alphabets $\mathcal{A}$ for synchronization actions and a finite set of real-valued variables $\mathcal{C}$ for clocks. We use a, b etc. to range over $\mathcal{A}$ and $x,\ y$ etc. to range over $\mathcal{C}$. Subsets of $\mathcal{C}$ will be denoted by $\mathbf{x}, \mathbf{y}$ with elements $x_i,\ x_j,\ \ldots,\ y_i,\ y_j,\ \ldots$. We use $\mathcal{B}(C)$, ranged over by ϕ, ψ etc., to denote the set of conjunctive formulas of atomic constraints of the form: $x_i \bowtie m$ or $x_i - x_j \bowtie n$ where $x_i, x_j \in \mathcal{C}$, $\bowtie \in \{\leq, <, \geq, >\}$, and m, n are natural numbers. The elements of $\mathcal{B}(C)$ are called *clock constraints*.

Definition 2.1. *A timed automaton over actions $\mathcal{A}$ and clocks $\mathcal{C}$ is a tuple $\langle N, l_0, E\rangle$ where*

- *N is a finite set of nodes,*
- *$l_0 \in N$ is the initial node,*
- *$E \subseteq N \times \mathcal{B}(C) \times \mathcal{A} \times 2^C \times N$ is the set of edges.*

When $\langle l, g, a, r, l'\rangle \in E$, we write $l \xrightarrow{g,a,r} l'$.

We shall present the operational semantics for timed automata in terms of a process algebraic language in which each term denotes an automaton.

Sometimes to describe progress properties, nodes of timed automata are associated with *invariants* that control the amount of time an automaton can stay at a node. Such an extension will be discussed in Section 5.

$$\text{DELAY}\quad \frac{}{t\rho \xrightarrow{d} t(\rho+d)}$$

$$\text{ACTION}\quad \frac{}{(a(\mathbf{x}).t)\rho \xrightarrow{a} t\rho\{\mathbf{x}:=0\}} \qquad \text{CHOICE}\quad \frac{t\rho \xrightarrow{a} t'\rho'}{(t+u)\rho \xrightarrow{a} t'\rho'}$$

$$\text{GUARD}\quad \frac{t\rho \xrightarrow{a} t'\rho'}{(\phi{\rightarrow}t)\rho \xrightarrow{a} t'\rho'}\ \rho \models \phi \qquad \text{REC}\quad \frac{(t[\mathbf{fix}Xt/X])\rho \xrightarrow{a} t'\rho'}{(\mathbf{fix}Xt)\rho \xrightarrow{a} t'\rho'}$$

Fig. 2. Standard Transitional Semantics

2.2 The Language

We preassume a set of process variables, ranged over by X, Y, Z, ... The language for timed automata over $\mathcal{C}$ can be given by the following BNF grammar:

$$t ::= \quad \mathbf{0} \quad | \quad \phi{\rightarrow}t \quad | \quad a(\mathbf{x}).t \quad | \quad t+t \quad | \quad X \quad | \quad \mathbf{fix}Xt$$

$\mathbf{0}$ is the inactive process which can do nothing, except for allowing time to pass. $\phi{\rightarrow}t$, read "if ϕ then t", is the usual (one-armed) conditional construct. $a(\mathbf{x}).r$ is action prefixing. $+$ is nondeterministic choice.

A recursion $\mathbf{fix}Xt$ binds X in t. This is the only binding operator in this language. It induces the notions of bound and free process variables as usual. Terms not containing free variables are *closed*. A recursion $\mathbf{fix}Xt$ is *guarded* if every occurrence of X in t is within the scope of an action prefixing.

The set of clock variables used in a term t is denoted $\mathcal{C}(t)$.

A *clock valuation* is a function from $\mathcal{C}$ to $\mathbf{R}^{\geq 0}$, and we use ρ to range over clock valuations. The notations $\rho\{\mathbf{x}:=0\}$ and $\rho+d$ are defined thus

$$\rho\{\mathbf{x}:=0\}(y) = \begin{cases} 0 & \text{if } y \in \mathbf{x} \\ \rho(y) & \text{otherwise} \end{cases}$$
$$(\rho+d)(x) = \rho(x)+d \quad \text{for all } x$$

Given a clock valuation $\rho : \mathcal{C} \to \mathbf{R}^{\geq 0}$, a term can be interpreted according to rules in Figure 2, where the symmetric rule for $+$ has been omitted. The transitional semantics uses two types of transition relations: action transition $\xrightarrow{a}$ and delay transition $\xrightarrow{d}$. We call $t\rho$ a *process*, where t is a term and ρ a valuation; we use p, q, ... to range over the set of processes. We also write μ for either an action or a delay (a real number).

Definition 2.2. *A symmetric relation R over processes is a timed bisimulation if $(p,q) \in R$ implies*

whenever $p \xrightarrow{\mu} p'$ then $q \xrightarrow{\mu} q'$ for some q' with $(p',q') \in R$.

We write $p \sim q$ if $(p,q) \in R$ for some timed bisimulation R.

$$\textsc{Action}\ \frac{}{a(\mathbf{x}).t \xrightarrow{\mathrm{tt},a,\mathbf{x}} t} \qquad \textsc{Choice}\ \frac{t \xrightarrow{b,a,r} t'}{t+u \xrightarrow{b,a,r} t'}$$

$$\textsc{Guard}\ \frac{t \xrightarrow{\psi,a,r} t'}{\phi{\to}t \xrightarrow{\phi\wedge\psi,a,r} t'} \qquad \textsc{Rec}\ \frac{t[\mathbf{fix}Xt/X] \xrightarrow{b,a,r} t'}{\mathbf{fix}Xt \xrightarrow{b,a,r} t'}$$

Fig. 3. Symbolic Transitional Semantics

The symbolic transitional semantics of this language is reported in Figure 3. Again the symmetric rule for + has been omitted. According to the symbolic semantics, each guarded closed term of the language gives rise to a timed automaton; On the other hand, it is not difficult to see that every timed automaton can be generated from a guarded closed term in the language. In the sequel we will use the phrases "timed automata" and "terms" interchangeably.

The two versions of transitional semantics can be related as follows:

Lemma 2.3. *1. If* $t \xrightarrow{\phi,a,\mathbf{x}} t'$ *then* $t\rho \xrightarrow{a} t'\rho\{\mathbf{x} := 0\}$ *for any* $\rho \models \phi$.

2. If $t\rho \xrightarrow{a} t'\rho'$ *then there exist* ϕ, $\mathbf{x}$ *such that* $\rho \models \phi$, $\rho' = \rho\{\mathbf{x} := 0\}$ *and* $t \xrightarrow{\phi,a,\mathbf{x}} t'$.

3 Constraints and Symbolic Bisimulation

This section is devoted to defining a symbolic version of timed bisimulation. To easy the presentation we shall fix two timed automata and symbolically, i.e. without evaluating clock constraints, compare them for bisimulation. To avoid clock variables of one automaton being reset by the other, we always assume the sets of clocks of the two timed automata under consideration are disjoint and write C for the union of the two clock sets. Let N be the largest natural number occurring in the constraints of the two automata. An atomic constraint over C with ceiling N has one of the two forms: $x \bowtie m$ or $x - y \bowtie n$ where $x, y \in C, \bowtie \in \{\leq, <, \geq, >\}$ and $m, n \leq N$ are natural numbers.

In the following, "atomic constraint" always means "atomic constraint over C with ceiling N". Note that given two timed automata there are only finite number of such atomic constraints. We shall use c to range over atomic constraints.

A constraint, or *zone*, is a boolean combination of atomic constraints. A constraint ϕ is consistent if there is some ρ such that $\rho \models \phi$. Let ϕ and ψ be two constraints. We write $\phi \Rightarrow \psi$ to mean $\rho \models \phi$ implies $\rho \models \psi$ for any ρ. Note that the relation $\Rightarrow$ is decidable.

A *region constraint*, or *region* for short, ϕ is a consistent constraint containing only the following atomic conjuncts:

- For each $i \in \{1, \ldots, n\}$ either $x_i = m_i$ or $m_i < x_i < m_i + 1$ or $x_i > N$;
- For each pair of $i, j \in \{1, \ldots, n\}$, $i \neq j$, either $x_i - m_i = x_j - m_j$ or $x_i - m_i < x_j - m_j$ or $x_i - m_i > x_j - m_j$.

where the m_i in $x_i - m_i$ of the second clause refers to the m_i related to x_i in the first clause. In words, m_i is the integral part of x_i and $x_i - m_i$ its fractional part.

Given a set of clock variables C and a ceiling N, the set of region constraints over C is finite. and is denoted $\mathcal{RC}_N^C$. In the sequel, we will omit the sub- and super-scripts when they can be supplied by the context.

Fact 1 *Let ϕ be a region constraint. If $\rho \models \phi$ and $\rho' \models \phi$ then*

- *For all $i \in \{1, \ldots, n\}$, if $\rho(x_i) \leq N$ then $\lfloor \rho(x_i) \rfloor = \lfloor \rho'(x_i) \rfloor$.*
- *For any $i, j \in \{1, \ldots, n\}$, $i \neq j$,*
  - *$\{\rho(x_i)\} = \{\rho(x_j)\}$ iff $\{\rho'(x_i)\} = \{\rho'(x_j)\}$ and*
  - *$\{\rho(x_i)\} < \{\rho(x_j)\}$ iff $\{\rho'(x_i)\} < \{\rho'(x_j)\}$.*

where $\lfloor x \rfloor$ and $\{x\}$ are the integral and fractional parts of x, respectively.

That is, two valuations satisfying the same region constraint must agree on their integral parts as well as on the ordering of their fractional parts. Note that this is precisely the definition of region equivalence due to Alur and Dill [AD94].

The notion of a region constraint enjoy an important property: processes in the same region behave uniformly with respect to timed bisimulation ([Cer92]):

Fact 2 *Let t, u be two timed automata with disjoint sets of clock variables and ϕ a region constraint over the union of the two clock sets. Suppose that both ρ and ρ' satisfy ϕ. Then $t\rho \sim u\rho$ iff $t\rho' \sim u\rho'$.*

Fact 3 *Suppose that ϕ is a region constraint and ψ a zone. Then either $\phi \Rightarrow \psi$ or $\phi \Rightarrow \neg\psi$.*

So a region is either entirely contained in a zone, or is completely outside a zone. In other words, regions are the finest polyhedra that can be described by our constraint language.

A *canonical* constraint is a disjunction of regions. Given a constraint we can first transform it into disjunctive normal form, then decompose each disjunct into a disjoint set of regions. Both steps can be effectively implemented. As a corollary to Fact 3, if we write $\mathcal{RC}(\phi)$ for the set of regions contained in the zone ϕ, then $\bigvee \mathcal{RC}(\phi) = \phi$, i.e. $\bigvee \mathcal{RC}(\phi)$ is the canonical form of ϕ.

We will need two operators to deal with resetting. The first one is $\downarrow_{\mathbf{x}}$ where $\mathbf{x} \subseteq C \subseteq \mathcal{C}$. We first define it on regions, then generalise it to zones. By the abuse of notation, we will write $c \in \phi$ to mean c is a conjunct of ϕ.

For a region ϕ,

$$\begin{aligned}\phi\downarrow_{\mathbf{x}} = \phi \downarrow'_{\mathbf{x}} \wedge\ & \bigwedge\{\, x_i = 0 \mid x_i \in \mathbf{x} \,\} \wedge \bigwedge\{\, x_i = x_j \mid x_i, x_j \in \mathbf{x} \,\} \\ \wedge\ & \bigwedge\{\, x_i = x_j - m \mid x_i \in \mathbf{x}, x_j \notin \mathbf{x}, x_j = m \in \phi \,\} \\ \wedge\ & \bigwedge\{\, x_i < x_j - m \mid x_i \in \mathbf{x}, x_j \notin \mathbf{x}, x_j > m \in \phi \,\}\end{aligned}$$

and $\downarrow'_{\mathbf{x}}$ is defined by

$$\begin{array}{ll}\mathsf{tt} \downarrow'_{\mathbf{x}} = \mathsf{tt} & \\ (c \wedge \phi) \downarrow'_{\mathbf{x}} = \phi \downarrow'_{\mathbf{x}} & \text{if } \mathbf{x} \cap fv(c) \neq \emptyset \\ (c \wedge \phi) \downarrow'_{\mathbf{x}} = c \wedge \phi \downarrow'_{\mathbf{x}} & \text{if } \mathbf{x} \cap fv(c) = \emptyset\end{array}$$

where $fv(c)$ is the set of clock variables appearing in (atomic constraint) c.

Lemma 3.1. *1. $\rho \models \phi$ implies $\rho\{\mathbf{x} := 0\} \models \phi\downarrow_{\mathbf{x}}$.*
2. If ϕ is a region constraint then so is $\phi\downarrow_{\mathbf{x}}$.

For a canonical constraint $\bigvee_i \phi_i$ with each ϕ_i a region, $(\bigvee_i \phi_i)\downarrow_{\mathbf{x}} = \bigvee_i(\phi_i\downarrow_{\mathbf{x}})$. For an arbitrary constraint ϕ, $\phi\downarrow_{\mathbf{x}}$ is understood as the result of applying $\downarrow_{\mathbf{x}}$ to the canonical form of ϕ.

The second operator $_\Uparrow$ is defined similarly. We first define it on regions:

$$\phi\Uparrow = \phi\Uparrow' \wedge \bigwedge_{i\neq j} e_{ij}(\phi)$$

where $_\Uparrow'$ is defined by

$$\begin{array}{l}
\mathsf{tt}\Uparrow' = \mathsf{tt} \\
(x < m \wedge \phi)\Uparrow' = x \leq N \wedge \phi\Uparrow' \\
(x = m \wedge \phi)\Uparrow' = m \leq x \wedge \phi\Uparrow' \\
(c \wedge \phi)\Uparrow' = c \wedge \phi\Uparrow' \quad \text{for other atomic constraint c}
\end{array}$$

and

$$e_{ij}(\phi) = \begin{cases} x_i - m_i = x_j - m_j & x_i = m_i, x_j = m_j \in \phi \\ m_i - m_j - 1 < x_i - x_j < m_i - m_j & \text{otherwise} \end{cases}$$

For an arbitrary constraint ϕ, $\phi\Uparrow$ is understood as the result of applying $\Uparrow$ to each disjunct of the canonical form of ϕ.

Definition 3.2. *ϕ is $\Uparrow$-closed if and only if $\phi\Uparrow = \phi$.*

Lemma 3.3. *1. $\phi\Uparrow$ is $\Uparrow$-closed.*
2. $\rho \models \phi$ implies $\rho \models \phi\Uparrow$.
3. If ϕ is $\Uparrow$-closed then $\rho \models \phi$ implies $\rho + d \models \phi$ for all $d \in \mathbf{R}^{\geq 0}$.

Symbolic bisimulation will be defined as a family of binary relations indexed by clock constraints. Following [Cer92] we use constraints over the union of the (disjoint) clock sets of two timed automata as indices. Given a constraint ϕ, a finite set of constraints Φ is called a *ϕ-partition* if $\bigvee \Phi = \phi$. A ϕ-partition Φ is called *finer* than another such partition Ψ if Φ can be obtained from Ψ by decomposing some of its elements. By the corollary to Fact 3, $\mathcal{RC}(\phi)$ is a ϕ-partition, and is the finest such partition. In particular, if ϕ is a region constraint then $\{\phi\}$ is the only partition of ϕ.

Definition 3.4. *A constraint indexed family of symmetric relations over terms $\mathbf{S} = \{ S^\phi \mid \phi \Uparrow\text{-closed} \}$ is a timed symbolic bisimulation if $(t, u) \in S^\phi$ implies*

> *whenever $t \xrightarrow{\psi, a, \mathbf{x}} t'$ then there is a $\phi \wedge \psi$-partition Φ such that for each $\phi' \in \Phi$ there is $u \xrightarrow{\psi', a, \mathbf{y}} u'$ for some ψ', $\mathbf{y}$ and u' such that $\phi' \Rightarrow \psi'$ and $(t', u') \in S^{\phi'\downarrow_{\mathbf{xy}}\Uparrow}$.*

We write $t \sim^{\phi} u$ *if* $(t,u) \in S^{\phi} \in \mathbf{S}$ *for some symbolic bisimulation* $\mathbf{S}$.

It is easy to see that the $\phi \wedge \psi$-partition Φ used in the above definition can be replaced by any partition finer than Φ.

Timed symbolic bisimulation captures $\sim$ in the following sense:

Theorem 3.5. $t \sim^{\phi} u$ *iff* $t\rho \sim u\rho$ *for any* $\rho \models \phi$.

Proof. ($\Longrightarrow$) Assume $(t,u) \in S^{\phi} \in \mathbf{S}$ for some symbolic bisimulation $\mathbf{S}$. Define

$$R = \{\, (t\rho, u\rho) \mid \text{there exists some } \phi \text{ such that } \rho \models \phi \text{ and } (t,u) \in S^{\phi} \in \mathbf{S} \,\}$$

We show R is a timed bisimulation. Suppose $(t\rho, u\rho) \in R$, i.e. there is some ϕ such that $\rho \models \phi$ and $(t,u) \in S^{\phi}$.

- $t\rho \xrightarrow{a} t'\rho'$. By Lemma 2.3 there are ψ, $\mathbf{x}$ such that $\rho \models \psi$, $\rho' = \rho\{\mathbf{x} := 0\}$ and $t \xrightarrow{\psi,a,\mathbf{x}} t'$. So there is a $\phi \wedge \psi$-partition Φ with the properties specified in Definition 3.4. Since $\rho \models \phi \wedge \psi$, $\rho \models \phi'$ for some $\phi' \in \Phi$. Let $u \xrightarrow{\psi',a,\mathbf{y}} u'$ be the symbolic transition associated with this ϕ', as guaranteed by Definition 3.4. Then $\phi' \Rightarrow \psi'$ and $(t',u') \in S^{\phi'\downarrow_{\mathbf{xy}}\Uparrow}$. Since $\rho \models \psi'$, $u\rho \xrightarrow{a} u'\rho\{\mathbf{y} := 0\}$. By Lemma 3.1, $\rho\{\mathbf{xy} := 0\} \models \phi'\downarrow_{\mathbf{xy}}$. By Lemma 3.3, $\rho\{\mathbf{xy} := 0\} \models \phi'\downarrow_{\mathbf{xy}}\Uparrow$. Therefore $(t'\rho\{\mathbf{xy} := 0\}, u'\rho\{\mathbf{xy} := 0\}) \in R$. Since $t'\rho\{\mathbf{xy} := 0\} \equiv t'\rho\{\mathbf{x} := 0\}$ and $u'\rho\{\mathbf{xy} := 0\} \equiv u'\rho\{\mathbf{y} := 0\}$, this is the same as $(t'\rho\{\mathbf{x} := 0\}, u'\rho\{\mathbf{y} := 0\}) \in R$.
- $t\rho \xrightarrow{d} t(\rho + d)$. Then also $u\rho \xrightarrow{d} u(\rho + d)$. Since ϕ is $\Uparrow$-closed, $\rho + d \models \phi$. Therefore $(t(\rho + d), u(\rho + d)) \in R$.

($\Longleftarrow$) Assume $t\rho \sim u\rho$ for any $\rho \models \phi_0$, we show $t \sim^{\phi_0} u$ as follows. For each $\Uparrow$-closed ϕ define

$$S^{\phi} = \{\, (t,u) \mid \forall \phi' \in \mathcal{RC}(\phi)\ \exists \rho \models \phi' \ s.t.\ (t\rho, u\rho) \in R \,\}$$

and let $\mathbf{S} = \{\, S^{\phi} \mid \phi \text{ is } \Uparrow - \text{closed} \,\}$ Then by Fact 2 $(t,u) \in S^{\phi_0}$. $\mathbf{S}$ is well-defined because R is a timed bisimulation. We show $\mathbf{S}$ is a symbolic bisimulation. Suppose $(t,u) \in S^{\phi}$ and let $t \xrightarrow{\psi,a,\mathbf{x}} t'$. Define $\Phi' = \{\, \phi' \mid \phi' \in \mathcal{RC}(\phi) \text{ and } \phi' \Rightarrow \psi \,\}$. Then Φ' is a $\phi \wedge \psi$-partition. For each $\phi' \in \Phi'$, there exists ρ s.t. $\rho \models \phi'$ with $(t\rho, u\rho) \in R$. By the definition of Φ', $\rho \models \psi$. By Lemma 2.3, $t\rho \xrightarrow{a} t'\rho\{\mathbf{x} := 0\}$. Since $(t\rho, u\rho) \in R$, $u\rho \xrightarrow{a} u'\rho'$ for some u' and ρ' with $(t'\rho\{\mathbf{x} := 0\}, u'\rho') \in R$. By Lemma 2.3 again, $u \xrightarrow{\psi',a',\mathbf{y}} u'$ for some ψ' and $\mathbf{y}$ with $\rho \models \psi'$ and $\rho' = \rho\{\mathbf{y} := 0\}$. Hence $(t\rho\{\mathbf{x} := 0\}, u\rho\{\mathbf{y} := 0\}) \in R$, which is the same as $(t\rho\{\mathbf{xy} := 0\}, u\rho\{\mathbf{xy} := 0\}) \in R$. ¿From $\rho \models \phi'$, by Lemma 3.1 we have $\rho\{\mathbf{xy} := 0\} \models \phi'\downarrow_{\mathbf{xy}}$. Since ϕ' is a region constraint, so is $\phi'\downarrow_{\mathbf{xy}}$ which is the only element of $\mathcal{RC}(\phi'\downarrow_{\mathbf{xy}})$. Therefore $(t',u') \in S^{\phi'\downarrow_{\mathbf{xy}}\Uparrow}$.

S1 $X + \mathbf{0} = X$
S2 $X + X = X$
S3 $X + Y = Y + X$
S4 $(X + Y) + Z = X + (Y + Z)$

Fig. 4. The Equational Axioms

4 The Proof System

The proof system consists of a set of equational axioms in Figure 4 and a set of inference rules in Figure 5. The judgments of the inference system are *conditional equations* of the form

$$\phi \rhd t = u$$

with ϕ a constraint and t, u terms. Its intended meaning is "$t\rho \sim u\rho$ for any $\rho \models \phi$". $t = u$ abbreviates $\mathsf{tt} \rhd t = u$.

The equational axioms are the standard monoid laws for bisimulation [Mil89]. The set of inference rules extends equational reasoning by introducing a rule for each construct in the process language. CONGR-+ expresses the fact that bisimulation is preserved by +. The rule GUARD permits a case analysis on conditional. It is all we need to reason with this construct. ACTION is the introduction rule for action prefixing. This rule is complicated by the fact that an action has associated with it a clock resetting, hence necessitates the two operators $\downarrow_{\mathbf{xy}}$ and $\Uparrow$. It requires a side condition to make sure clock resetting in one process does not interfere with the other. Finally, the two rules PARTITION and ABSURD have nothing to do with any specific constructs in the language. They are so-called "structural rules" used to "glue" pieces of derivations together.

Let us write $\vdash \phi \rhd t = u$ to mean $\phi \rhd t = u$ can be derived from this proof system.

Some useful properties of the proof system are summarised in the following proposition:

Proposition 4.1. *1.* $\vdash \phi{\rightarrow}(\psi{\rightarrow}t) = \phi \wedge \psi{\rightarrow}t$
2. $\vdash t = t + \phi{\rightarrow}t$
3. If $\phi \Rightarrow \psi$ *then* $\vdash \phi \rhd t = \psi{\rightarrow}t$
4. $\vdash \phi \wedge \psi \rhd t = u$ *implies* $\vdash \phi \rhd \psi{\rightarrow}t = \psi{\rightarrow}u$
5. $\vdash \phi{\rightarrow}(t + u) = \phi{\rightarrow}t + \phi{\rightarrow}u$

The rule PARTITION has a more general form:

Proposition 4.2. *Suppose* Φ *is a finite set of constraints and* $\bigvee \Phi = \phi$. *If* $\vdash \psi \rhd t = u$ *for each* $\psi \in \Phi$, *then* $\vdash \phi \rhd t = u$.

Soundness of the proof system is stated below:

Theorem 4.3. *If* $\vdash \phi \rhd t = u$ *and* ϕ *is* $\Uparrow$*-closed then* $t \sim^{\phi} u$.

Now we discuss the completeness of the proof system, and we shall confine to the recursion-free subset of the language. As usual the completeness proof uses the notion of a normal form.

EQUIV $\quad \dfrac{}{t = t} \quad \dfrac{t = u}{u = t} \quad \dfrac{t = u \quad u = v}{t = v}$

AXIOM $\quad \dfrac{}{t = u}$ $\quad t = u$ is an axiom instance

CONGR-+ $\quad \dfrac{t = t'}{t + u = t' + u}$

GUARD $\quad \dfrac{\phi \wedge \psi \rhd t = u \quad \phi \wedge \neg\psi \rhd \mathbf{0} = u}{\phi \rhd \psi{\rightarrow}t = u}$

ACTION $\quad \dfrac{\phi{\downarrow_{\mathbf{xy}}}{\Uparrow} \rhd t = u}{\phi \rhd a(\mathbf{x}).t = a(\mathbf{y}).u} \quad \mathbf{y} \cap \mathcal{C}(t) = \mathbf{x} \cap \mathcal{C}(u) = \emptyset$

PARTITION $\quad \dfrac{\phi_1 \rhd t = u \quad \phi_2 \rhd t = u}{\phi \rhd t = u} \quad \phi \Rightarrow \phi_1 \vee \phi_2$

ABSURD $\quad \dfrac{}{\mathsf{ff} \rhd t = u}$

Fig. 5. The Inference Rules

Definition 4.4. *A term t is a normal form if $t \equiv \sum_i \phi_i{\rightarrow}a_i(\mathbf{x}_i).t_i$ and each t_i is a normal form.*

Definition 4.5. *The height of a term t, denoted $\mid t \mid$, is defined thus:*

- $\mid \mathbf{0} \mid = 0$
- $\mid t + u \mid = \ max\{\mid t \mid, \mid u \mid\}$
- $\mid \phi{\rightarrow}t \mid = \mid t \mid$
- $\mid a(\mathbf{x}).t \mid = 1 + \mid t \mid$

Lemma 4.6. *For every term t there exists a normal form t' such that $\mid t \mid = \mid t' \mid$ and $\vdash t = t'$.*

Theorem 4.7. *For recursion-free terms t and u, $t \sim^{\phi} u$ implies $\vdash \phi \rhd t = u$.*

Proof. By Lemma 4.6 we assume $t,\ u$ are in normal form:

$$t \equiv \sum_{i \in I} \phi_i{\rightarrow}a_i(\mathbf{x}_i).t_i$$
$$u \equiv \sum_{j \in J} \psi_j{\rightarrow}b_j(\mathbf{y}_j).u_j$$

Without loss of generality, we may assume $a_i = b_j = a$ for all i and j.

Apply induction on the joint height of t and u. The base case is trivial. For the induction step, let $\phi' \in \mathcal{RC}(\phi)$. For each $i \in I$, $t \stackrel{\phi_i, a, \mathbf{x}_i}{\longrightarrow} t_i$. Since $t \sim^{\phi} u$, there exists a $\phi \wedge \phi_i$-partition Φ with the properties specified in Definition 3.4.

Without lose of generality, we assume each element of Φ is a region constraint, i.e. $\Phi = \mathcal{RC}(\phi \wedge \phi_i)$. Since ϕ' is a region, by Fact 3 there are two cases:
(case 1) $\phi' \wedge \phi_i = \mathsf{ff}$, i.e. $\phi' \notin \Phi$. By GUARD and ABSURD we can derive $\vdash \phi' \rhd \phi_i \rightarrow a(\mathbf{x}_i).t_i = \mathbf{0}$.
(case 2) $\phi' \Rightarrow \phi_i$, i.e. $\phi' \in \Phi$. By the definition of symbolic bisimulation, there is some $j \in J$ such that $\phi' \Rightarrow \psi_j$, $u \xrightarrow{\psi_j, a, \mathbf{y}_j} u_j$ and $t_i \sim^{\phi' \downarrow_{\mathbf{x}_i \mathbf{y}_j} \Uparrow} u_j$. By induction we have

$$\vdash \phi' \downarrow_{\mathbf{x}_i \mathbf{y}_j} \Uparrow \rhd t_i = u_j$$

By ACTION,

$$\vdash \phi' \rhd a(\mathbf{x}_i).t_i = a(\mathbf{y}_j).u_j$$

Since $\phi' \Rightarrow \phi_i$ and $\phi' \Rightarrow \psi_j$, by Proposition 4.1,

$$\vdash \phi' \rhd \phi_i \rightarrow a(\mathbf{x}_i).t_i = \psi_j \rightarrow a(\mathbf{y}_j).u_j$$

Symmetrically, for each $j \in J$, either $\vdash \phi' \rhd \psi_j \rightarrow a(\mathbf{y}_j).u_j = \mathbf{0}$ or there is some $i \in I$ such that

$$\vdash \phi' \rhd \psi_j \rightarrow a(\mathbf{y}_j).u_j = \phi_i \rightarrow a(\mathbf{x}_i).t_i$$

Therefore, using S1 – S4 and CONGR-+, we can conclude

$$\vdash \phi' \rhd t = \sum_{i \in I} \phi_i \rightarrow a_i(\mathbf{x}_i).t_i + \sum_{j \in J} \psi_j \rightarrow b_j(\mathbf{y}_j).u_j$$

and

$$\vdash \phi' \rhd u = \sum_{i \in I} \phi_i \rightarrow a_i(\mathbf{x}_i).t_i + \sum_{j \in J} \psi_j \rightarrow b_j(\mathbf{y}_j).u_j$$

Hence

$$\vdash \phi' \rhd t = u$$

Finally an application of Proposition 4.2 gives the required

$$\vdash \phi \rhd t = u$$

5 Invariants

One important variation on the notion of timed automata is to associate an invariant condition to each node of the automaton to model progress behaviours. According to the transitional semantics of Figure 2 a process can delay forever at any location (node). To disallow such arbitrary delays each location in a timed automata is assigned an *invariant constraint*, with the interpretation that delay transitions at a node will not be possible when the invariant at the node is violated.

To describe timed automata with invariants we extend our language as follows

$$\begin{array}{lll} s ::= & \{\phi\}t & \\ t ::= & \mathbf{0} \mid \phi \rightarrow t \mid a(\mathbf{x}).s \mid t + t \mid X \mid \mathbf{fix} X t & \end{array}$$

In this language, invariants can not occur at places which do not correspond to locations in timed automata. For instance, strings having the forms $\phi{\rightarrow}\{\psi\}t$, $\{\phi\}t + \{\psi\}u$ or $\mathbf{fix}X\{\phi\}t$ are *not* terms of the language, while $\{\phi\}(t+u)$ and $\phi{\rightarrow}a(\mathbf{x}).\{\psi\}t$ are allowed.

We assign each term t an invariant constraint $Inv(t)$ by letting

$$Inv(t) = \begin{cases} \phi & \text{if } t \text{ has the form } \{\phi\}t' \\ \mathsf{tt} & \text{otherwise} \end{cases}$$

Furthermore, we add a side condition to the rule DELAY in Figure 2, plus a new rule INV to deal with invariants:

$$\textsc{delay}\ \frac{}{t\rho \xrightarrow{d} t(\rho+d)}\ \ \rho + d' \models Inv(t) \text{ for any } 0 \le d' \le d$$

$$\textsc{inv}\ \ \frac{t\rho \xrightarrow{a} t'\rho'}{(\{\phi\}t)\rho \xrightarrow{a} t'\rho'}\ \ \rho \models \phi$$

For the symbolic transitional semantics, we simply forget the invariants (recall that symbolic transitions correspond to edges of automata, while invariants reside in nodes):

$$\textsc{Inv}\ \frac{t \xrightarrow{\psi,a,r} t'}{\{\phi\}t \xrightarrow{\psi,a,r} t'}$$

A slightly modified version of Lemma 2.3 holds:

Lemma 5.1. *1. If $t \xrightarrow{\phi,a,\mathbf{x}} t'$ then $t\rho \xrightarrow{a} t'\rho\{\mathbf{x}:=0\}$ for any $\rho \models \phi \wedge Inv(t)$.*
2. If $t\rho \xrightarrow{a} t'\rho'$ then there exist ϕ, $\mathbf{x}$ such that $\rho \models \phi \wedge Inv(t)$, $\rho' = \rho\{\mathbf{x}:=0\}$ and $t \xrightarrow{\phi,a,\mathbf{x}} t'$.

Definition 2.2 of timed bisimulation remains the same, since the intended effects of invariants have already been manifested in the the transition rules for the standard transitional semantics. On the other hand, the definition of symbolic timed bisimulation, Definition 3.4, should be modified slightly to accommodate invariants:

Definition 5.2. *A constraint indexed family of symmetric relations over terms $\mathbf{S} = \{ S^\phi \mid \phi \Uparrow\text{-closed}\}$ is a timed symbolic bisimulation if $(t,u) \in S^\phi$ implies*

1. *$\phi \Rightarrow (Inv(t) \Rightarrow Inv(u))$ and*
2. *whenever $t \xrightarrow{\psi,a,\mathbf{x}} t'$ then there is a $Inv(t) \wedge \phi \wedge \psi$-partition Φ such that for each $\phi' \in \Phi$ there is $u \xrightarrow{\psi',a,\mathbf{y}} u'$ for some ψ', $\mathbf{y}$ and u' such that $\phi' \Rightarrow \psi'$ and $(t',u') \in S^{\phi'\downarrow_{\mathbf{xy}}\Uparrow}$.*

We write $t \sim^\phi u$ if $(t,u) \in S^\phi \in \mathbf{S}$ for some symbolic bisimulation $\mathbf{S}$.

We have the counterpart of Theorem 3.5:

Theorem 5.3. *$t \sim^{\phi} u$ iff $t\rho \sim u\rho$ for any $\rho \models \phi$.*

The proof of this theorem is very similar to that of Theorem 3.5, with the uses of Lemma 2.3 replaced by Lemma 5.1.

Concerning the proof system, we add a rule to deal with the construct $\{\phi\}t$:

$$\text{INV}\ \frac{\phi \wedge \psi \rhd t = u \qquad \phi \wedge \neg\psi \rhd \{\mathsf{ff}\}\mathbf{0} = u}{\phi \rhd \{\psi\}t = u}$$

This rule appears similar to the GUARD rule. However, there is a crucial difference: When the guard ψ is false $\psi{\rightarrow}t$ behaves like $\mathbf{0}$, the process which is inactive but can allow time to pass; On the other hand, when the invariant ψ is false $\{\psi\}t$ behaves like $\{\mathsf{ff}\}\mathbf{0}$, the process usually referred to as *time-stop*, which is not only inactive but also "still", can not even let time elapse.

With these modifications the completeness result carries over to the new setting:

Theorem 5.4. *For recursion-free terms t and u in the extended language, $t \sim^{\phi} u$ implies $\vdash \phi \rhd t = u$.*

The proof uses the following normal form taking invariants into account:

$$\{\psi\} \sum_{i \in I} \phi_i{\rightarrow}a_i(\mathbf{x}_i).t_i$$

The technical details of the proof are almost the same as that of Theorem 4.7.

6 Conclusion

We have proposed a theory of symbolic bisimulation and presented a proof system for timed automata. Using conditional equations as judgments the proof system separates manipulation of time from reasoning about process equivalence. As a result the proof system is much simpler than purely equational formulation. It is shown that by generalising pure equational reasoning to a set of inference rules dealing with specific language constructs needed for timed automata, the standard monoid laws for bisimulation are sufficient for characterizing bisimulation in the timed world. This result agrees with the previous works on proof systems for value-passing processes [HL96] and for π-calculus [Lin94], providing a further evidence that the four monoid laws capture the essence of bisimulation.

The proof system presented in the current paper is complete only over finite timed automata, i.e. the subset of timed automata which do not involve loops. We conjecture that by adding a suitable version of *unique fixpoint induction* [Mil84], together with the standard laws for folding/unfolding recursions, a complete proof system for the whole set of timed automata can be achieved. A similar result has been reported in [AJ94] for regular timed CCS [Wan91]. We leave this as a topic for future research.

References

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

[AJ94] L. Aceto and A. Jeffrey. A complete axiomatization of timed bisimulation for a class of timed regular behaviours. Report 4/94, Sussex University, 1994.

[Bor96] M. Boreale. Symbolic Bisimulation for Timed Processes. In *AMAST'96*, LNCS 1101 pp.321-335. Springer–Verlag. 1996.

[Cer92] K. Čeräns. Decidability of Bisimulation Equivalences for Parallel Timer Processes. In *CAV'92*, LNCS 663, pp.302-315. Springer–Verlag. 1992.

[DAB96] P.R. D'Argenio and Ed Brinksma. A Calculus for Timed Automata (Extended Abstract). In *FTRTFTS'96*, LNCS 1135, pp.110-129. Springer–Verlag. 1996.

[HL95] M. Hennessy and H. Lin. Symbolic bisimulations. *Theoretical Computer Science*, 138:353–389, 1995.

[HL96] M. Hennessy and H. Lin. Proof systems for message-passing process algebras. *Formal Aspects of Computing*, 8:408–427, 1996.

[Lin94] H. Lin. Symbolic bisimulations and proof systems for the π-calculus. Report 7/94, Computer Science, University of Sussex, 1994.

[Mil84] R. Milner. A complete inference system for a class of regular behaviours. *J. Computer and System Science*, 28:439–466, 1984.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[Wan91] Wang Yi. *A Calculus of Real Time Systems*. Ph.D. thesis, Chalmers University, 1991.

[WPD94] Wang Yi, Paul Pettersson, and Mats Daniels. Automatic Verification of Real-Time Communicating Systems By Constraint-Solving. In *Proc. of the* 7th *International Conference on Formal Description Techniques*, 1994.