A Proof System for Timed Automata

Huimin Lin
Wang Yi

Department of Information Technology
Uppsala University

Technical report 1999-013
November 1999
ISSN 1404-3203

A Proof System for Timed Automata

Huimin Lin
Laboratory for Computer Science
Inst. of Software, Chinese Academy of Sciences
Email: lhm@ox.ios.ac.cn

Wang Yi
Dept. 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 over the recursion-free subset of the language. The completeness proof relies on the notion of *symbolic timed bisimulation*. Two variations of the axiomatisation are also discussed, one on timed automata by associating an invariant constraint to each node and the other on bisimulation by abstracting away delay transitions.

1 Introduction

Timed automata [AD94] has emerged as a popular model for formal analysis and verification of real time systems. By now theories of timed automata have been well developed, but there is still one aspect missing: axiomatisation.

Timed automata extend the traditional (finite state) labelled transition graphs by associating a finite set of real-valued *clock variables* (or just *clocks* for short) to each graph and annotating each transition with, in addition to an action label, a time constraint and a subset of clocks. An automaton may stay at a node with clocks increasing uniformly (to model time passage), or choose a transition whose time 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 processes [HL95, HL96] to the timed setting. To this end we first present a simple CCS-style language and

associate a timed automaton to each term in the language by means of *symbolic transitional semantics*. 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 $a(\overline{x}).t$, meaning to perform action a and reset the clocks in $\overline{x}$ to zero, then behave like t. An inference system is then formulated whose judgments are *conditional equations* of the form

$$\phi \rhd t = u$$

Intuitively it says "t and u are timed bisimilar over clock valuations satisfying ϕ". A typical inference rule takes the form:

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

It performs a case analysis on the constraint ϕ: $\phi{\rightarrow}t$ behaves like t when ϕ is true, and like the inactive process $\mathbf{0}$ otherwise. Note that the guarding constraint ϕ of $\phi{\rightarrow}t$ 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:

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

Here $\downarrow_{\overline{xy}}$ and $\uparrow$ are postfixing operations on clock constraints. $\phi{\downarrow_{\overline{xy}}}{\uparrow}$ is a time constraint obtained from ϕ by first setting the clocks in $\overline{xy}$ to zero (operator $\downarrow_{\overline{xy}}$), 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 $\downarrow_{\overline{xy}}$ 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 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 u are symbolically bisimilar over indexing constraint ϕ if and only if they are timed bisimilar for any time valuation satisfying ϕ.

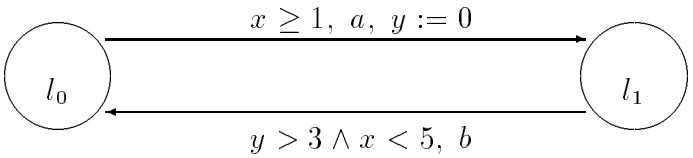

Figure 1: A Timed Automaton.

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 a timed automaton to each term in the language. Section 3 develops a theory of symbolic bisimulation for timed automata. The proof system is presented in Section 4, together with its completeness proof. Two variations are discussed in Section 5. The paper is concluded with Section 6 where further research direction is also outlined.

Related work The only previous attempt to axiomatizing timed automata we are aware is [DAB96]. A process algebra style language was proposed for timed automata, and a large set of equations for timed bisimulation were put forward. The soundness of the axiomatisation was demonstrated. However, no completeness result was 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 an 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 condition on clocks), a *synchronisation action*, and a *clock reset* (a subset of clocks to be reset). Intuitively, a timed automaton starts execution with all clocks set to zero. Clocks increase uniformly with time while the automaton is within a node. A transition can be taken if the clocks fulfill the guard. By taking the transition, all clocks in the

clock reset will be set to zero, while the remaining keep their values. 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 $(l, < s, t >)$, 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 $(l_0, < 0, 0 >)$, 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 $(l_0, < t, t >)$ with $t \geq 0$ are reachable. The edges of a timed automaton are decorated with a condition (guard) on the clock values that must be satisfied in order to be enabled. Thus, only at the states (l_0, t, t), where $t \geq 1$, the edge from l_0 to l_1 is enabled. Additionally, edges are be 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 $(l_1, < t, 0 >)$, 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 $\mathcal{C}$. 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,*
- *$E \subseteq N \times \mathcal{B}(C) \times \mathcal{A} \times \in^{\mathcal{C}} \times \mathcal{N}$ 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 5.1.

$$\text{DELAY}\ \frac{}{t\rho \xrightarrow{d} t(\rho+d)} \qquad \text{ACTION}\ \frac{}{(a(\overline{x}).r)\rho \xrightarrow{a} r\rho\{\overline{x}:=0\}} \qquad \text{CHOICE}\ \frac{t\rho \xrightarrow{a} t'\rho'}{(t+u)\rho \xrightarrow{a} t'\rho'}$$

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

Figure 2: Standard Transitional Semantics

2.2 The Language

Assume 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:

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

$\mathbf{0}$ is the inactive process which is capable of doing nothing. $\phi{\rightarrow}t$ is the (one-armed) conditional construct. $a(\overline{x}).r$ 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*. Note that because of the way the syntax is defined, every process variable in a term necessarily occurs within the scope of a action prefixing, *i.e.* all terms are *guarded*.

The set of clocks 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 $\rho\{\overline{x}:=0\}$ and $\rho+d$ are defined thus

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

Given a clock valuation $\rho : \mathcal{C} \to \mathbf{R}^{\geq 0}$, a term can be interpreted according to rules in Figure 2. The transitional semantics uses two kind 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 delay.

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

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

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

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

Figure 3: Symbolic Transitional Semantics

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

The symbolic transitional semantics of this language is reported in Figure 3, where the symmetric rule for + has been omitted. By the symbolic semantics, each 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 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 $t \xrightarrow{\phi,a,\overline{x}} t'$ then $t\rho \xrightarrow{a} t'\rho\{\overline{x} := 0\}$ for any $\rho \models \phi$.*

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

3 Constraints and Symbolic Bisimulation

Let C be a set of clock variables and N a natural number. An atomic constraint over C with ceiling N has one of the following forms:

$$x \prec m, \quad x \succ m, \quad x - y \prec m, \quad x - y \succ m$$

where $\prec \in \{\leq, <\}$, $x \in C$, $\succ \in \{\geq, >\}$, $x \in C$ and $m \leq N$.

In the sequel "atomic constraint" always means "atomic constraint over C with ceiling N". As we are interested in proving bisimulation equivalence between two timed automata, ceiling N is taken to be the largest constant appearing in the two automata under consideration. Note that given two timed automata there are only finite number of 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 $x_i - m_i > x_j - m_j$.

Given a set of clock variables $\overline{x} \subseteq \mathcal{C}$ and a ceiling N, the set of region constraints over $\overline{x}$ is finite and is denoted $\mathcal{RC}_N(\overline{x})$.

Fact 3.1 *Let ϕ be a region constraint. If $\rho \models \phi$ and $\rho' \models \phi$ then ($\lfloor x \rfloor$ and $\{x\}$ are the integral and fractional parts of x, respectively)*

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

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.

Fact 3.2 *Suppose that ϕ is a region constraint and ψ a zone. Then either $\phi \Rightarrow \psi$ or $\phi \wedge \psi = false$.*

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. In the sequel we will not distinguish between a constraint and its canonical form, i.e. we assume every constraint is already in canonical form.

We will need two operators to deal with resetting. The first one is $\downarrow_{\overline{x}}$ where $\overline{x} \subseteq \mathcal{C}$. We first define it on regions, then generalise it to zones.

For a region ϕ,

$$\begin{array}{rcl} \phi\downarrow_{\overline{x}} = \phi\downarrow'_{\overline{x}} & \wedge & \bigwedge\{\, x_i = 0 \mid x_i \in \overline{x}\,\} \wedge \bigwedge\{\, x_i = x_j \mid x_i, x_j \in \overline{x}\,\} \\ & \wedge & \bigwedge\{\, x_i = x_j - m \mid x_i \in \overline{x}, x_j \notin \overline{x}, x_j = m \text{ is a conjunct of } \phi\,\} \\ & \wedge & \bigwedge\{\, x_i < x_j - m \mid x_i \in \overline{x}, x_j \notin \overline{x}, x_j > m \text{ is a conjunct of } \phi\,\} \end{array}$$

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

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

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

Lemma 3.3 *1. $\rho \models \phi$ implies $\rho\{\overline{x} := 0\} \models \phi\downarrow_{\overline{x}}$.*

2. If ϕ is a region constraint then so is $\phi{\downarrow_{\overline{x}}}$.

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

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

$$\begin{array}{l} true\uparrow = true \\ (x < m \wedge \phi)\uparrow = \phi\uparrow \\ (c \wedge \phi)\uparrow = c \wedge \phi\uparrow \ \text{ for other atomic constraint c} \end{array}$$

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 3.4 ϕ is $\uparrow$-closed if and only if $\phi\uparrow = \phi$. □

Lemma 3.5 *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 $d \in \mathbf{R}^{\geq 0}$.

Now we are ready to define symbolic 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. Symbolic bisimulation will be defined as a family of binary relations indexed by clock constraints. Following [Cer92] we use constraints over the union of clock variables of the two timed automata as indexes.

Definition 3.6 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 $t \xrightarrow{\psi,a,\overline{x}} t'$ then there is a $\phi \wedge \psi$-partition Φ such that for each $\phi' \in \Phi$ there is $u \xrightarrow{\psi',a,\overline{y}} u'$ for some ψ', $\overline{y}$ and u' such that $\phi' \Rightarrow \psi'$ and $(t',u') \in S^{\phi'{\downarrow_{\overline{xy}}}\uparrow}$.

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

Theorem 3.7 *$t \sim^\phi u$ 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 $(t,u) \in S^\phi$.

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

Figure 4: The Equational Axioms

- $t\rho \xrightarrow{a} t'\rho'$. By Lemma 2.3 there are ψ, $\overline{x}$ such that $\rho \models \psi$, $\rho' = \rho\{\overline{x} := 0\}$ and $t \xrightarrow{\psi,a,\overline{x}} t'$. So there is a $\phi \wedge \psi$-partition Φ with the properties specified in Definition 3.6. Since $\rho \models \phi \wedge \psi$, $\rho \models \phi'$ for some $\phi' \in \Phi$. Let $u \xrightarrow{\psi',a,\overline{y}} u'$ be the symbolic transition associated with this ψ', as guaranteed by Definition 3.6. Then $\phi' \Rightarrow \psi'$ and $(t', u') \in S^{\phi'\downarrow_{\overline{xy}}\uparrow}$. Since $\rho \models \psi'$, $u\rho \xrightarrow{a} u'\rho\{\overline{y} := 0\}$. By Lemma 3.3, $\rho\{\overline{xy} := 0\} \models \phi'\downarrow_{\overline{xy}}$. By Lemma 3.5, $\rho\{\overline{xy} := 0\} \models \phi'\downarrow_{\overline{xy}}\uparrow$. Therefore $(t'\rho\{\overline{xy} := 0\}, u'\rho\{\overline{xy} := 0\}) \in R$. Since $t'\rho\{\overline{xy} := 0\} \equiv t'\rho\{\overline{x} := 0\}$ and $u'\rho\{\overline{xy} := 0\} \equiv u'\rho\{\overline{y} := 0\}$, this is the same as $(t'\rho\{\overline{x} := 0\}, u'\rho\{\overline{y} := 0\}) \in R$.

- $t\rho \xrightarrow{d} t(\rho + d)$. 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 $t\rho \sim u\rho$, i.e. $(t\rho, u\rho) \in R$ for some timed bisimulation R. For each $\uparrow$-closed ϕ define

$$S^{\phi} = \{\, (t, u) \mid \begin{array}{l} \text{there is a set of region constraints } \Phi \text{ } s.t. \text{ } \bigvee \Phi = \phi \\ \text{and } \forall \phi' \in \Phi \ \exists \rho \models \phi' \text{ with } (t\rho, u\rho) \in R \end{array} \,\}$$

and let $\mathbf{S} = \{S^{\phi}\}$. $\mathbf{S}$ 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 $t \xrightarrow{\psi,a,\overline{x}} t'$. Let Φ be the set of region constraints required in the definition of S^{ϕ}. Define $\Phi' = \{\, \phi' \mid \phi' \in \Phi \text{ and } \phi' \Rightarrow \phi \wedge \psi \,\}$. Then Φ' is a $\phi \wedge \psi$-partition. For each $\phi' \in \Phi'$, there exists ρ s.t. $\rho \models \phi'$ with $(t\rho, u\rho) \in R$. Hence $\rho \models \psi$. By Lemma 2.3, $t\rho \xrightarrow{a} t'\rho\{\overline{x} := 0\}$. Since $(t\rho, u\rho) \in R$, $u\rho \xrightarrow{a} u'\rho'$ for some u' and ρ' with $(t'\rho\{\overline{x} := 0\}, u'\rho') \in R$. By Lemma 2.3 again, $u \xrightarrow{\psi',a',\overline{y}} u'$ for some ψ' and $\overline{y}$ with $\rho \models \psi'$ and $\rho' = \rho\{\overline{y} := 0\}$. Hence $(t\rho\{\overline{xy} := 0\}, u\rho\{\overline{xy} := 0\}) \in R$. ¿From $\rho \models \phi'$, by Lemma 3.3 we have $\rho\{\overline{xy} := 0\} \models \phi'\downarrow_{\overline{xy}}$. Since ϕ' is a region constraint, so is $\phi'\downarrow_{\overline{xy}}$. Hence $(t', u') \in S^{\downarrow_{\overline{xy}}\uparrow}$. □

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 judgements of the inference system are *conditional equations* of the form

$$\phi \rhd t = u$$

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

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

ABSURD $\quad \dfrac{}{\mathit{false} \rhd t = u}$

Figure 5: The Inference Rules

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 $\mathit{true} \rhd t = u$.

The equational axioms are the standard monoid laws for bisimulation [Mil89]. The set of inference rules extends equational reasoning by introducing an 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 ${\downarrow_{\overline{xy}}}$ 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 $\phi \models \psi$ *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 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.

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

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

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

□

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 $t \sim^{\phi} u$ *implies* $\vdash \phi \rhd t = u$.

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

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

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

$$\Phi = \{\, \phi' \mid \phi' \in \mathcal{RC}(\mathcal{C}(t) \cup \mathcal{C}(u)),\ \phi' \Rightarrow \phi \,\}$$

Fix a $\phi' \in \Phi$. For each $i \in I$, $t \stackrel{\phi_i, a, \overline{x}_i}{\longrightarrow} t_i$. Since $t \sim^{\phi} u$, there exists a $\phi \wedge \phi_i$-partition Φ' with the properties specified in Definition 3.6. Since ϕ' is a region constraint, by Fact 3.2 there are two cases:
(case 1) $\phi' \wedge \phi_i = \mathit{false}$. Then $\Phi' = \emptyset$ and we have $\vdash \phi' \rhd \phi_i \rightarrow a(\overline{x}_i).t_i = \mathbf{0}$.
(case 2) $\phi' \Rightarrow \phi_i$. Then $\phi' \wedge \phi = \phi'$ and $\Phi' = \{\phi'\}$. By the deiniftion of symbolic bisimulation, there is some $j \in J$ such that $\phi' \Rightarrow \psi_j$, $u \stackrel{\psi_j, a, \overline{y}_j}{\longrightarrow} u_j$ and $t_i \sim^{\phi' \downarrow_{\overline{x}_i \overline{y}_j} \uparrow} u_j$. By induction we have

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

By ACTION,

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

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

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

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

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

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

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

Finally an application of Proposition 4.2 gives

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

□

5 Some Variations

There are some variations on the notions of timed automata and bisimulation. In this section we will briefly discuss two of them: one on timed automata by associating an invariant constraint to each node, the other on bisimulation by abstracting away delay transitions.

5.1 Invariants

According to the transitional semantics of Figure 2 a process can delay forever at any location (node). To disallow such arbitrary delays one can associate an *invariant constraint* to each location, with the interpretation that delay transitions at a node will not be possible when the invariant at the node is violated. Formally we add one clause

$$t ::= \ldots \quad | \quad \{\phi\}t$$

to the language, and define a function Inv, mapping a term to a constraint, by the following rules:

$$\begin{array}{ll} Inv(\mathbf{0}) = true & Inv(a(\overline{x}).t) = true \\ Inv(t+u) = Inv(t) \wedge Inv(u) & Inv(t \parallel u) = Inv(t) \wedge Inv(u) \\ Inv(\psi \rightarrow t) = true & Inv(\{\phi\}t) = \phi \wedge Inv(t) \\ Inv(X) = true & Inv(\mathbf{fix}Xt) = Inv(t) \end{array}$$

Furthermore, add a side condition to the rule DELAY in Figures 2:

$$\text{DELAY} \quad \frac{}{t\rho \xrightarrow{d} t(\rho + d)} \quad \rho + d' \models Inv(t) \text{ for any } 0 \leq d' \leq d$$

Definition 2.2 of timed bisimulation remains the same, since the intended effects of invariants have already been manifested in the side condition of the above rule. On the other hand, the definition of symbolic timed bisimulation, Definition 3.6, should be modified slightly to accommodate invariants:

Definition 5.1 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 $t \xrightarrow{\psi,a,\overline{x}} t'$ then there is a $Inv(t) \wedge Inv(u) \wedge \phi \wedge \psi$-partition Φ such that for each $\phi' \in \Phi$ there is $u \xrightarrow{\psi',a,\overline{y}} u'$ for some ψ', $\overline{y}$ and u' such that $\phi' \Rightarrow \psi'$ and $(t',u') \in S^{\phi'\downarrow_{\overline{xy}}\uparrow}$.

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

Concerning the proof system, add a new rule INV to take invariants into count:

$$\text{INV} \quad \frac{\psi \wedge \phi \rhd t = u}{\psi \rhd \{\phi\}t = u}$$

With these modifications the main results obtained in the previous two sections carry over to the new setting:

Theorem 5.2 *$t \sim^{\phi} u$ iff $t\rho \sim u\rho$ for any $\rho \models Inv(t) \wedge Inv(u) \wedge \phi$.*

Theorem 5.3 *$t \sim^{\phi} u$ implies $\vdash Inv(t) \wedge Inv(u) \wedge \phi \rhd t = u$.*

The proofs of the about two theorems are very similar to their counterparts in the previous sections and are omitted.

5.2 Untimed Bisimulation

If we only require two timed automata to match action transitions and do not insist on they match delay transitions, we arrive at *untimed (or time-abstract) bisimulation.* Let time-abstract transition relation $\Longrightarrow$ be defined thus : $p \stackrel{a}{\Longrightarrow} q$ if and only if $p \stackrel{d}{\longrightarrow}\stackrel{a}{\longrightarrow} q$ for some $d \in \mathbf{R}^{\geq 0}$.

Definition 5.4 A symmetric relation R between processes is an untimed bisimulation if $(p,q) \in R$ implies

> whenever $p \stackrel{a}{\Longrightarrow} p'$ then $q \stackrel{a}{\Longrightarrow} q'$ for some q' and $(p',q') \in R$.

We write $p \approx q$ if $(p,q) \in R$ for some untimed bisimulation R. □

To obtain a symbolic characterisation of untimed bisimulation, we only need to change $(t',u') \in S^{\phi'\downarrow_{\overline{xy}}\uparrow}$ in Definition 3.6 to $(t',u') \in S^{\phi'\downarrow_{\overline{x}}\downarrow_{\overline{y}}\uparrow}$, thus allow two processes not to match time elapses.

Definition 5.5 A constraint indexed family of symmetric relations over terms $\mathbf{S} = \{\, S^{\phi} \mid \phi \uparrow\text{-closed} \,\}$ is a untimed symbolic bisimulation if $(t,u) \in S^{\phi}$ implies

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

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

Theorem 5.6 *$t \approx^{\phi} u$ iff $t\rho \approx u\rho$ for any $\rho \models \phi$.*

In line with the modification for symbolic bisimulation, to obtain a proof system for untimed bisimulation, all needed is to replace the rule ACTION in Figure 5 with the following

$$\text{ACTION-UT} \quad \frac{\phi\downarrow_{\overline{x}}\downarrow_{\overline{y}}\uparrow \rhd t = u}{\phi \rhd a(\overline{x}).t = a(\overline{y}).u} \quad \mathcal{C}(t) \cap \mathcal{C}(u) = \emptyset$$

With these modifications we have the completeness theorem for the untimed version:

Theorem 5.7 *$t \approx^{\phi} u$ implies $\vdash \phi \rhd t = u$.*

The proof of this theorem is almost identical to that of Theorem 4.7, with the use of ACTION replaced by ACTION-UT.

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 enjoys nice modularity in the sense that variations on either automata or bisimulation equivalence can be easily handled by adding/modifying relevant rules. It is shown that by generalising pure equational reasoning to a system 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], and provides a further evidence that the four monoid laws capture the essence of bisimulation.

The proof system reported in the current paper is complete only over finite timed automata, i.e. the subset of timed automata that 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 achieve. 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.