Testing Deadlock-freeness in Real-time Systems; A Formal Approach

Behzad BORDBAR[1] and Kozo OKANO[2]

[1] University of Birmingham B.Bordbar@cs.bham.ac.uk
[2] Osaka University okano@ist.osaka-u.ac.jp

Abstract. A *Time Action Lock* is a state of a Real-time system at which neither time can progress nor an action can occur. Time Action Locks are often seen as signs of errors in the model or inconsistencies in the specification. As a result, finding out and resolving Time Action Locks is a major task for the designers of Real-time systems. Verification is one of the methods of discovering deadlocks. However, due to state explosion, the verification of deadlock freeness is computationally expensive. The aim of this paper is to present a computationally cheap testing method for Timed Automata models and pointing out any source of *possible* Time Action Locks to the designer.
We have implemented the approach presented in the paper, which is based on the geometry of Timed Automata, via a Testing Tool called TALC (Time Action Lock Checker). TALC, which is used in the conjunction with the model checker UPPAAL, tests the UPPAAL model and provides feedback to the designer. We have illustrated our method by applying TALC to a model of a simple communication protocol.

Keywords: Testing, Real-time System, Deadlock, Timed Automata, Rational Presburger Sentences, Communication Protocol

1 Introduction

In a general term, a deadlock is a state at which a system is unable to progress any further. Various types of deadlock in Real-time systems are studied in the literature [16, 8, 7, 27, 28]. In particular, a *Time Lock* [27] is a state at which time is prevented from passing beyond a certain point, and *Time Action Lock* [8] is a Time Lock state at which no action can occur. As a result, a Time Action Lock, is a state at which neither time can progress nor an action can occur.

In this paper, we shall deal with Real-time systems, which are modelled via Timed Automata [1]. Such systems can be verified with the help of model checkers such as UPPAAL [2, 6], which uses a variant of Timed Automata model of [1]. UPPAAL has been successfully applied to the verification of Real-time systems [5, 15, 20, 9, 2].

The process of verification of a property σ starts by creating a UPPAAL Timed Automata model of the Real-time system. Before conducting the verification of the property σ, we often check the model for the existence of deadlocks. This is to ensure the integrity of the design; as the existence of a deadlock is often interpreted as either an error in the model or a sign of inconsistencies in the specification. As a result, when a model checker informs us of the existence of a deadlock, we scrutinise the model to discover the cause of the deadlock. However, due to state explosion, the verification of deadlock freeness is computationally expensive. The aim of this paper is to present a method of testing of the Timed Automata models to point out any source of *possible* Time Action Locks to the designer. This is to help avoiding the verification of the model for deadlock-freeness, which is computationally expensive. Our approach can be implemented via a *Testing Tool*, which works in parallel with a model checker as depicted in Fig. 1. The *designer* creates a model of the system in the *Model Checker*. The Testing Tool *checks* the model for Time Action Locks and provides feedback to the designer. The feedback provided to the designer is either, *"the system is deadlock free"* or *"there is a possibility of deadlocks."* In the case that the system is declared deadlock free by the Testing Tool, there is no need to use the Model Checker to ensure the system is deadlock free, and the designer can focus on the verification of σ. If the Testing tool declares that there is a possibility of deadlocks, sources of the deadlock are pointed out, which can help the designer in scrutinising the model for finding any possible flaw in the model or inconsistencies in the specification.

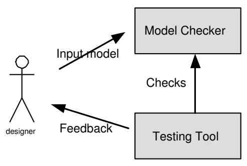

Fig. 1. Combining Testing tool and Model Checker

The approach presented in this paper is based on the geometry of the Timed Automata. In a Timed Automaton, the progress of time is subject to a set of constraints, which form convex regions [27] in the n-dimensional Eucleadian space $\mathbb{R}^n$. As a result, for every location of a Timed Automaton, various types of constraint such as *invariants* and *guards* correspond to regions in $\mathbb{R}^n$. The idea behind our approach is to identify subsets of such regions that might cause a Time Action Lock and test them.

Based on our approach, we have developed a Testing Tool called *Time Action Lock Checker* (TALC). TALC, which works in conjunction with UPPAAL, tests the Timed Automata via Rational Presburger Sentences and is available for download at `http://www.cs.bham.ac.uk/~bxb/TALC.html`.

The paper is organised as follows. We shall start by a brief introduction on the Timed Automata. Section 3 follows with a brief review of the background material on Presburger Arithmetic. Section 4 reviews definitions of various types of Time Lock. Section 5 sketches our geometric approach for detecting Time Action Lock. Results related to the implementation via Rational Presburger Sentences are discussed in section 6. Section 7 explains the Testing Tool TALC and applies the method to the testing of a simple communication protocol for the existence of a Time Action Lock. The paper finishes with a conclusion section.

2 Timed Automata

In this section, we shall review a variation of Timed Automata model proposed by Alur and Dill [1], which is used in UPPAAL [2, 6, 21], a tool for the verification of behavioural properties of Real-time systems.

Consider $\mathcal{X} = \{x_1, \ldots, x_n\}$ a set of clock variables with values in $\mathbb{R}_+$, the set of non-negative real numbers. Suppose that $\mathbf{c}_1(\mathcal{X})$ is the set of all constraints created from conjunctions of atomic formals of the form $x_i \sim q$, where $x_i \in \mathcal{X}$, $\sim \in \{\leq, \geq, <, >, =\}$ and $q \in \mathbb{Q}_+$, the set of non-negative Rational numbers. Also, assume that $\mathbf{c}_2(\mathcal{X})$ is the set of all constraints created from the conjunction of atomic formula's of the form $x_i \sim q$ and $x_i - x_j \sim q$, where x_i, c and $\sim$ are as above and for $i \neq j$, $x_j \in \mathcal{X}$. The set of all possible constraints is defined by $\mathbf{c}(\mathcal{X}) = \mathbf{c}_1(\mathcal{X}) \cup \mathbf{c}_2(\mathcal{X})$. We shall refer to $x_i \sim q$ and $x_i - x_j \sim q$, as *atomic* constraints. [3]

A *valuation* (*variable assignment*) is a map $v : \mathcal{X} \to \mathbb{R}_+$, which assigns to each clock a non-negative Real-number. For a valuation v, a delay $d \in \mathbb{R}_+$, which is denoted by $v + d$, is defined as $(v + d)(x) = v(x) + d$, if $x \in \mathcal{X}$. In other words, all clocks operate with the same speed. Let $\mathscr{V}(A)$ denote the set of all valuations.

The value of clock can be *reset*. A reset statement is of the form $x := e$, where $x \in \mathcal{X}$. In the current version of UPPAAL, e must be an integer. A set of reset statements is called a *reset-set* or *reset* if each variable is assigned at most once. The result of applying a reset r to a valuation v is denoted by the valuation $r(v)$. If a variable x is such that no assignment of r changes its value then $v(x) = r(v)(x)$. Let $\mathscr{R}$ denotes the set of all resets.

A *Timed Automaton* $\mathscr{A}$ is a 6-tuple $(L, l_0, T, I, \mathcal{X}, init, A)$ such that

- $L = \{l_0, \ldots, l_N\}$ is a finite set of *locations* and $l_0 \in L$ is a designated location called the *initial location.* Assume that $init(l_0) \in \mathbf{c}(\mathcal{X})$ assigns to the initial location an *initial region*.
- $\mathcal{X}$ and A are finite sets of clock variables and actions, respectively.
- $T \subset L \times A \times \mathbf{c}(\mathcal{X}) \times \mathscr{R} \times L$ is the set of transition relation. An element of T is of the form of (l_i, a, g, r, l_j), where $l_i, l_j \in L$ and $a \in A$ is an action, $g \in \mathbf{c}(\mathcal{X})$ is called a *guard*, and $r \in \mathscr{R}$ is a set of reset statements. We sometimes write $l_i \xrightarrow{a,g,r} l_j$ to depict that $\mathscr{A}$ evolves from a location l_i to a new location l_j, if the guard g is evaluated *true*, the action a is performed and clocks and data variables are reset according to r. In this case, we shall refer to $e = (a, g, r)$ as the edge connecting l_i and l_j. We shall also write $\mathsf{Action}(e)$, $\mathsf{guard}(e)$ and $\mathsf{reset}(e)$ to denote a, g, and r, respectively.
- $I : L \to \mathbf{c}(\mathcal{X})$ is a function that assigns to each location an *invariant*. Intuitively, a Timed Automata can stay in a location while its invariants are satisfied. The default invariant for a location is *true* $(x \geq 0)$.

Notation 1 *For a location $l \in L$, we shall write $^\circ l$ to denote the set of all edges $e = (a, g, r)$ ending in l. Similarly, l° denotes the set of all edges starting from l.*

[3] The Timed Automata model of UPPAAL contains both clock variables, as defined above, and *data variables*, which have integer values. In order to simplify our model, we shall only be dealing with clock variables. Also, the current version of TALC only implements the clock variables.

The semantics of Timed Automata can be interpreted over transition systems, *i.e.* triple $(S, s_0, \Rightarrow)$, where

- $S \subset L \times \mathscr{V}$ is the set of *states*, i.e. each state is a pair (l, v), where l is a location and v is a valuation
- $s_0 \in S$ is an *initial state*, and
- $\Rightarrow \subset S \times (A \cup \mathbb{R}_+) \times S$ is a *transition relation*, where A is the set of *all* actions.

A transitions can be either a *discrete transitions*, e.g. (s_1, a, s_2), where $a \in A$ or a *time transitions*, *e.g.* (s_1, d, s_2), where $d \in \mathbb{R}_+$ and denotes the passage of d time units. Transitions are written: $s_1 \overset{e}{\Rightarrow} s_2$ and $s_1 \overset{d}{\Rightarrow} s_2$, respectively, and are defined according of the following inference rules:

$$\frac{l_1 \xrightarrow{a,g,r} l_2, g(v)}{(l_1, v) \overset{e}{\Rightarrow} (l_2, r(v))} \quad \frac{\forall d' \leq d \;\; I(l)(v + d')}{(l, v) \overset{d}{\Rightarrow} (l, v + d)}$$

To model concurrency and synchronisation between Timed Automaton, CCS [22] style parallel composition operators are introduced, which synchronise over half actions. We refer the interested reader to [2] for further details of the UPPAAL model of Timed Automata.

Assume that $\mathscr{A}$ is a Timed Automaton. A *run* σ of $\mathscr{A}$ is a finite/infinite sequence of transitions of the form $s_0 \overset{\lambda_1}{\Rightarrow} s_1 \overset{\lambda_2}{\Rightarrow} s_2 \cdots$ where s_0 is the initial state and $\lambda_i \in A \cup \mathbb{R}_+$, where A is the set of actions. For further information on network of Timed Automata and UPPAAL see [6, 21].

3 Rational Presburger Sentences

Assume that F denotes the set of all linear inequalities on integer variables and integer constants. A *Presburger Sentence* is a closed first-order logical statements on F. The phrase *closed* means that, there is no free variable in a Presburger Sentence. For example, $\forall x \exists y (3x + 7 \leq y \vee y \geq 0)$ is a Presburger Sentence. Satisfiability of Presburger Sentence is decidable [19].

A *Rational Presburger Sentence* (RPS) is similar to the conventional Presburger Sentences, except that constants are rational numbers and variables range over rational (or real) numbers. As a result, the syntax of RPS is as follow:

$\mathsf{fact} ::= x \mid ax \mid b$, where x is a rational-valued variable and a and b are integer (or rational) constants

$\mathsf{exp} ::= \mathsf{fact} \mid \mathsf{fact} + \mathsf{exp} \mid \mathsf{fact} - \mathsf{exp}$

$\mathsf{lterm} ::= \mathsf{exp} = \mathsf{exp} \mid \mathsf{exp} > \mathsf{exp} \mid \mathsf{exp} < \mathsf{exp}$

$\mathsf{lexp} ::= \mathsf{lterm} \mid \neg\mathsf{lexp} \mid \mathsf{lexp} \wedge \mathsf{lterm}$

$\mathsf{RPS} ::= \forall x\ \mathsf{lexp}$, where x is any free variable in lexp.

Note that, for RPS F and G, $\exists x F$ can be defined as $\neg \forall x \neg F$ and $F \vee G$ as $\neg(\neg F \wedge \neg G)$. Also $f \geq g$ and $f \leq g$ are defined $f > g \vee f = g$ and $f < g \vee f = g$, respectively. The decision problem for RPS is decidable [13, 24]. Moreover, the computational times for deciding the satisfiability for RPS is less than that of Presburger Sentences. RPS and original Presburger Sentences have been successfully applied to the verification of logical designs and network design protocols [10, 14, 3, 25, 4]. Tools [23, 24] are available for the verification of RPS and Presburger sentences.

4 Deadlock in Timed Automata

Deadlocks, which have often been seen as error situations in concurrent and distributed systems, are classically interpreted as states at which the system will never be able to perform an *action*. In a Timed Automaton, a deadlock can *also* be created by preventing the passing of timed beyond a certain point, i.e. the elapse of time causes a violation of at least one of the constraints of the system. This situation, which is referred to as a *Timelock,* is often created as a result of fault in the specification of guards or invariant in the model. Finding out and resolving Timelocks is a major problem for the analysis and design of time critical systems.

Various interpretations of deadlock are extensively studied in the literature [8, 7, 27, 16, 28]. There are two different forms of Timelock [8], *Zeno Timelock* and *Time Action Lock.* Zeno Timelock is the case that infinite number of actions are performed in a finite period of time. This paper is about Time Action Lock, which is defined as follows in [8].
Time-Action-Lock A *Time-Action-Lock* (TAL) is a state at which time can *only* progress for a finite amount $0 < d < \infty$ of time but no action can occur.
A special case of the above definition is the situation at which, there is a reachable state at which neither time can progress nor an action can occur [8].

Example 1. Fig. 2 depicts two Timed Automata with TAL at the location l_0. The Left Hand Side Timed Automata has a TAL at the state $(l_0, 10)$, as neither time can progress, because of the violation of the invariant $x \leq 10$, nor an action can occur, because of the violation of the guard $8 \leq x < 9$. In both Timed Automata the reachable state $(l_0, 9)$ is also a TAL, as time can pass $d = 1$ unit and no action can occur. However, $(l_0, 10)$ is not a TAL state of the Right Hand Side Timed Automaton, as it is not a reachable state.

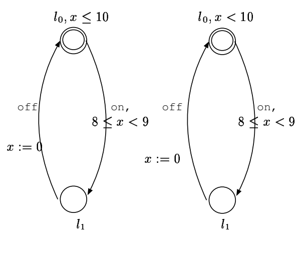

Fig. 2. Timed Automata with Time-Action-Lock

5 A Geometric Approach to the detection of Time-Action-Locks

A valuation is a function assigning real values to clocks $\mathcal{X} = \{x_1, \ldots, x_n\}$. As a result, we can identify a valuation $v : \mathcal{X} \to \mathbb{R}_+$ as a vector $\alpha = (\alpha_1, \ldots, \alpha_n)$, where for each $1 \leq i \leq n$, $\alpha_i = v(x_i)$. Hence, for each constraint $c \in \mathbf{c}(\mathcal{X})$ there is a subset of $\mathbb{R}_+^n$ of all points that satisfy c. We shall refer to such subset of $\mathbb{R}_+^n$ as the *corresponding region* of c and denote it with $\underline{c}$. *For the rest of this section assume that* $c \in \mathbf{c}(\mathcal{X})$ *and* $\underline{c}$ *denotes the corresponding region.*

Definition 1. *Assume that* $n \geq 1$. *For* $x \in \mathbb{R}_+^n$, *we shall write* $\Gamma(x, \underline{c}) = \sup\{t \geq 0 \mid x + t \times \mathbf{1} \in \underline{c}\}$,[4] *where* $\mathbf{1} = (1, \ldots, 1) \in \mathbb{R}_+^n$. *If there is a t such that* $x + t \times \mathbf{1} \in \underline{c}$ *and* $\Gamma(x, \underline{c}) < \infty$, *we shall write Fringe* $(x, \underline{c}) = x + \Gamma(x, \underline{c}) \times \mathbf{1}$ *and call it the* Fringe of x *with respect to* $\underline{c}$. *If* $\Gamma(x, \underline{c}) = \infty$, *we shall write Fringe*$(x, \underline{c}) = \{\infty\}$. *If there is no t such that* $x + t \times \mathbf{1} \in \underline{c}$, *then* $\Gamma(x, \underline{c}) = -\infty$, *and Fringe* $(x, \underline{c}) = \{-\infty\}$.

Example 2. Suppose that $n = 2$. Assume that a region c is specified by the conjunction of $x_1 \leq 1$, $x_2 \leq 2$, $x_1 \geq 0$ and $x_2 \geq 0.5$. Let $x = (0.5, 1)$, it can be seen in Fig. 3 that $\Gamma(x, \underline{c}) = 0.5$ and $\text{Fringe}(x, \underline{c}) = (1, 1.5)$.

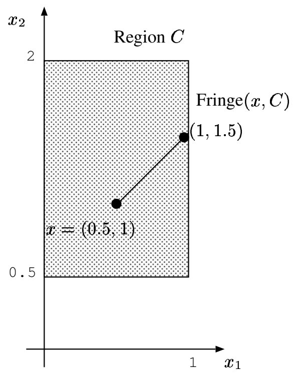

Fig. 3. Example 2

If we assume that the region $\underline{c}$ denotes the invariant of a location l of a Timed Automata $\mathscr{A}$. For a reachable state (l, x), we have $x \in \underline{c}$, where $c = g(l)$, where $\underline{c}$ denotes the corresponding region. In this case, $\Gamma(x, \underline{c})$ is the maximum amount of time that can expire while the location remains in l.

Definition 2. *If* $c_1, c_2 \in \mathbf{c}(\mathcal{X})$ *and* $\underline{c_1} \subset \underline{c_2}$, *then define Fringe*$(\underline{c_1}, \underline{c_2}) = \cup_{x \in \underline{c_1}}$ *Fringe*$(x, \underline{c_2})$.

[4] sup stands for *Supremum*. For each $A \subset \mathbb{R}$, $\sup A$ is the least upper bound of A. For example, $\sup[0, 1] = \sup[0, 1) = 1$. Each nonempty bounded subset of $\mathbb{R}$ has a supremum and the supremum of a nonempty unbounded subset of $\mathbb{R}$ is ∞. The supremum of empty set is defined as $-\infty$.

Assume that $c \in \mathbf{c}(\mathcal{X})$, then $\underline{c} \subset \mathbb{R}_+^n$ denotes the corresponding region. Assume that $\overline{\underline{c}}$ denotes the Topological Closure of $\underline{c}$ [11]. The reader, who is not familiar with the notion of Topological Closure, can use the following instead of the definition of the Topological Closure.

Lemma 1. *Suppose that $c_1 \in C(\mathcal{X})$ and c_2 is created from c_1 by replacing all $<$ $(>)$ with $\leq$ $(\geq)$, respectively. Then $\underline{c_2}$ is the Topological Closure of $\underline{c_1}$, i.e. $\underline{c_2} = \overline{(\underline{c_1})} = \overline{\underline{c_1}}$.*

Proof. By induction on the number of atomic constraints in c_1 and considering that the Topological Closure of the union of finite sets is the union of the Topological Closure.

The next theorem, in a layman language, states that a Timed Automaton is Time-Action-Lock free, if the fringe of the invariants of each location with respect to the guards of incoming transitions are covered by the Topological Closure of the guard of the outgoing transitions.

Theorem 1. *Assume that $\mathscr{A}$ is a Timed Automaton. $\mathscr{A}$ is Time-Action-Lock free if for each location l of $\mathscr{A}$,*

$$Fringe(A_l, B_l) \subset \overline{C_l}, \tag{1}$$

where

- *$A_l = \underline{\mathsf{init}(l)}$, if l is the initial location;*
- *if l is not an initial location or l is the initial [5] location with $^\circ l \neq \emptyset$, then $A_l = \bigcup_{e \in {}^\circ l} A_{l,e}$, in which $A_{l,e} = \underline{\mathsf{guard}(e)} \cap \underline{\mathsf{inv}(l)} \cap \underline{\mathsf{reset}(e)}$, where $\underline{\mathsf{reset}(e)}$ is the area of $\mathbb{R}_+^n$ corresponding the reset set of e, i.e. $\{x \in \mathbb{R}_+^n \mid x_i = 0 \text{ for } x_i \in \mathsf{reset}(e)\}$;*
- *$B_l = \underline{\mathsf{inv}(l)}$ is the region corresponding to the invariant of l;*
- *$C_l = \bigcup_{e \in l^\circ} C_{l,e}$, where $C_{l,e} = \underline{\mathsf{guard}(e)} \cap \underline{\mathsf{inv}(l)}$;*
- *$\overline{C_l}$ denotes the Topological Closure of C_l.*

Proof. Assume that for all locations l the equation 1 is satisfied, but $\mathscr{A}$ has a Time-Action-Lock. As a result, there is Time-Action-Lock state (l, x). We shall prove that the above assumption results in a contradiction. Without any loss of generality, we can assume that $x \in A_l$. This is because, there is a reachable state (l, z), where $z \in A_l$, such that by elapse of $0 < d < \infty$ unit of time ends in (l, x) and we can state the proof for (l, z). [To see this, assume that $\sigma = s_0 \overset{\lambda_1}{\Rightarrow} \cdots \overset{\lambda_n}{\Rightarrow} s_n = (l, x)$ is a run of $\mathscr{A}$ starting at an initial state and ending in (l, x). Suppose that there exists an action transition in the set $\{\lambda_1, \ldots, \lambda_n\}$, and assume that s_i is the last state before s_n at which an action transition occurs. Then, there is an edge $e \in {}^\circ l$ such that λ_i is the action of e. After $\lambda_{i+1} + \ldots + \lambda_n$ unit of time, the state (l, x) is reached. If there is no action transition in $\{\lambda_1, \ldots, \lambda_n\}$, simply let (l, z) to be the initial state s_0 and the time elapse of $d = \sum \lambda_i$ ends in the state (l, x). So let us assume that $x \in A_l$.]

By the definition of Time-Action-Lock at the state (l, x) time can elapse *only* $0 \leq d < \infty$ units and no action can occur. As a result $\gamma = \sup\{t \mid x + t \times \mathbf{1} \in B_l\} < \infty$ exists. Let $y = x + \gamma \times \mathbf{1}$ =Fringe(x, B_l). By equation (1) $y \in \overline{C_l}$. As a result, there

[5] If $^\circ l \neq \emptyset$ the initial location is studied twice, once with $A_l = \underline{\mathsf{init}(l)}$ and the second time like any other location.

are two cases; either y is and *interior* point or a boundary point. **case 1:** $y \in C_l$, By the definition of C_l, there exists $e \in l^\circ$ such that $y \in \mathsf{guard}(e) \cap \mathsf{inv}(l)$. Hence at state (l, x) time can elapse γ units to the new state (l, y) and then e can occur. This contradicts with (l, x) is a Time-Action-Lock state. **case 2:** $y \in \overline{C_l} \backslash C_l$, *i.e.*, y is in the Topological boundary of C_l. By the definition of the Topological boundary, each neighbourhood of y contains a point of C_l. Consider a point which also belongs to the line joining x to y. Then there is $0 < t < \gamma$ such that $x + t \times \mathbf{1} \in C_l$. Hence, there is $e \in l^\circ$ such that $x + t \times \mathbf{1} \in \mathsf{guard}(e) \cap \mathsf{inv}(l)$. This is a contradiction with (l, x) is a Time-Action-Lock as at state (l, x), time can pass t units to the new state $(l, x + t \times \mathbf{1})$ and then e occurs. As a result, in both cases, assuming that $\mathscr{A}$ has a TAL, results in a contradiction. $///$

Notice that the condition presented in the lemma is a necessary condition. In other words, if Equation (1) satisfies the Timed Automaton has no Time-Action-Lock. We argue that violation of equation 1, which *may* result in a Time-Action-Lock, is a sign of bad design. Based on this idea, we have developed a tool that carries a static analysis of the Timed Automata and points out to the designer any potential Time-Action-Lock. We shall explain our approach with the help of an example.

Example 3. Consider the Timed Automaton of Fig. 4, which models a switch on/off system. Fig. 5 depicts $A_{l1} = \mathsf{guard}(e1) \cap \mathsf{inv}(l1) \cap \mathsf{reset}(e1) = \{x \in \mathbb{R}^2_+ \mid x_2 = 0, 0 \leq x_1 \leq 2\}$, $B_{l1} = \mathsf{inv}(l1) = \{x \in \mathbb{R}^2_+ \mid x_2 \leq 1\}$ and $C_{l1} = \mathsf{guard}(e2) \cap \mathsf{inv}(l1) = \{x \in \mathbb{R}^2_+ \mid x_2 \leq 1 \, and \, 2 \leq x_1\}$.

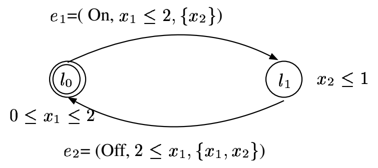
e_1=(On, $x_1 \leq 2$, $\{x_2\}$)
l_0
l_1
$x_2 \leq 1$
$0 \leq x_1 \leq 2$
e_2= (Off, $2 \leq x_1$, $\{x_1, x_2\}$)

Fig. 4. A Timed Automata with Time Action Lock

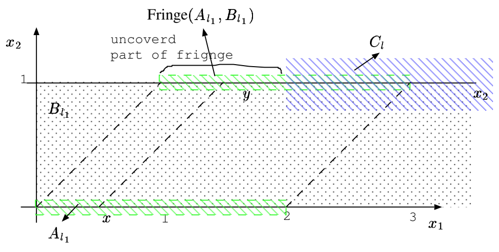

Fig. 5. Part of the Fringe is not covered

The next section presents a method of calculating the *Fringe*. Here, as depicted in Fig. 5, a direct use of the definition 1 shows that Fringe $(A_{l_1}, B_{l_1}) = \{x \in \mathbb{R}_+^2 \mid x_2 = 1, 2 \leq x_1 \leq 3\}$. It can clearly be seen that the part of Fringe (A_{l1}, B_{l1}) lying on $x_2 = 1$ with $x_1 < 2$ is not covered. The Fig. 5 demonstrates the idea behind the above Theorem. Clearly, at any state (l_1, x), where $x = (x_1, x_2)$ and $0 \leq x_1 \leq 1$ and $x_2 = 0$, if the time elapses by 1 unit, the state (l, y) is reached. At this point neither time can pass, since $\mathsf{inv}(l_1)$ gets violated, nor an action can occur, since no guard of an outgoing transition is satisfied. We can argue that the case presented in the above examples a clear case of wrong specification. In other words, there is a clear inconsistence in the specification that must be corrected. The contrary position, as explained in [8], is that such "error situations in behavioural techniques should have a behavioural/operational intuition that is justifiable in term of real world behaviour." This paper does not address the above hotly debated views. Our aim is to present a computationally cheap method of discovering such situations and pointing them to the designer.

Remark: The method presented in this paper deals *only* with a single Timed Automaton. As Bowman [8] points out, a Time Action Lock can *also* be created from unsuitable parallel composition. We are currently working on extending our method to cover *networks of Timed Automata*, i.e. parallel composition of Timed Automata. The current implementation of TALC, checks a *network of Timed Automata* only by studying each individual Timed Automaton component.

6 Applying Rational Presburger Sentences to Detect TAL

In this section, we shall present a method of detecting potential Time-Action-Lock (TAL) using Theorem 1. Considering equation 1 of Theorem 1, the aim is to present a technique to verify statements of the form Fringe$(\underline{c_1}, \underline{c_2}) \subset \underline{c_3}$, where c_1, c_2 and $c_3 \in c(\mathcal{X})$ and $\underline{c_1} \subset \underline{c_2}$. We shall verify such statements via Presburger sentences. However, first we shall present a set of results which facilitate the translation to suitable Rational Presburger Sentences, with minimal amount of computation.

Proposition 1. *For each $c \in \mathbf{c}(\mathcal{X})$ the corresponding $\underline{c}$ region is a convex set. Moreover, if $c \in \mathbf{c}_1(\mathcal{X})$, then $\underline{c}$ is a rectangular region, i.e. a Cartesian product of intervals.*

Proof. The convexity of the region is proved in [27]. The second part is by induction on the number of atomic formulas in c.

Notation 2 *Assume that $\underline{c}$ is a rectangular region of the form $I_1 \times \cdots \times I_n$, where each I_i is a (non-negative) real line interval. Then by Top Corner, we mean the point $(\alpha_1, \ldots, \alpha_n)$, where each α_i is the end point of I_i. Notice, in case of an unbounded interval, $\alpha_i = \infty$. Also, define the Bottom Corner to be the point $(\beta_1, \ldots, \beta_n)$, where each β_i is the starting point of the interval I_i.*

Calculating the Fringe for a rectangular region is straight forward.

Lemma 2. *If $c \in \mathbf{c}_1(\mathcal{X})$, i.e. $\underline{c}$ is a rectangular region, and $(\alpha_1, \ldots, \alpha_n)$ denotes the Top Corner point of the rectangular region $\underline{c}$, then for each point $x = (x_1, \ldots, x_n) \in \underline{c}$,*

– $\mathit{\Gamma}(x, \underline{c}) = \min\{\alpha_i - x_i \mid 1 \leq i \leq n\}$ *and*
– *Fringe*$(x, \underline{c}) = x + \min\{\alpha_i - x_i \mid 1 \leq i \leq n\} \times \mathbf{1}$.

Proof. If for each i, $\alpha_i = +\infty$, then $\mathit{\Gamma}(x, \underline{c}) = \infty$ and there is nothing to prove. Assume that for at least one co-ordinate t, $\alpha_t < \infty$. Since α_i is the end point of the interval coordinate of $\underline{c}$, $x + t \times \mathbf{1} \in \underline{c}$, if and only if for each coordinate i, $x_i + t \sim \alpha_i$, where $\sim \in \{\leq, <\}$. As a result, $\mathit{\Gamma}(x, \underline{c}) = \sup\{t \mid t \sim \alpha_i - x_i \text{ for } 1 \leq i \leq n\}$, which is the same as $\min\{\alpha_i - x_i \mid 1 \leq i \leq n\}$.

It might seem that, the above lemma, which provide an elegant way of computing the Fringe is only applicable to the rectangular regions. However, the following lemma shows that to calculate the Fringe, we only need to discard conditions of the form $x_i - x_j \sim q$ and focus on the rectangular regions.

Lemma 3. *If $\underline{c}$ is a non-empty region created from a constraint in $c \in \mathbf{c}_2(\mathcal{X})$. Let $c' \in \mathbf{c}(\mathcal{X})$ is a constraint created from modifying c by cancelling all atomic formulas of the form $x_i - x_j \sim q$. Also, assume that $\underline{c'}$ is the region (rectangular region) corresponding to c'. Then, $\underline{c} \subset \underline{c'}$ and for $x \in \underline{c}$ $\quad$ Fringe$(x, \underline{c})$ =Fringe$(x, \underline{c'})$.*

Proof. Since c' is created from the relaxing conditions of c, we have $\underline{c} \subset \underline{c'}$. For each $x \in \underline{c}$, we shall prove that $\sup\{t \mid x + t \times \mathbf{1} \in \underline{c}\} = \sup\{t \mid x + t \times \mathbf{1} \in \underline{c'}\}$. To see this, let LS denotes the left hand side supremum and RS denotes the right hand side supremum. Since $\underline{c} \subset \underline{c'}$, we get $LS \leq RS$. If the two are not equal by the definition of the supremum, there is a t such that $x + t \times \mathbf{1} \notin \underline{c}$ and $x + t \times \mathbf{1} \in \underline{c'}$. This can only happen because there is a condition of the form of $x_i - x_j \sim q$ that the vector $x + t \times \mathbf{1}$, does not satisfy i.e. the inequality $(x_i - t) - (x_j - t) \sim q$ is not satisfied. But, this is impossible, as $(x_i - t) - (x_j - t) = x_i - x_j$ and $x_i - x_j \sim q$.

In other words, to calculate the Fringe, we can ignore constraints of the form $x_i - x_j \sim q$ and focus on the rectangular regions. The following result identifies a less complex set that embodies the Fringe of a point.

Lemma 4. *If $\underline{c}$ is a non-empty region created from a constraint in $c \in \mathbf{c}_1(\mathcal{X})$ and $\alpha = (\alpha_1, \ldots, \alpha_n)$ is the Top Corner of $\underline{c}$. For $x \in \underline{c}$, Fringe$(x, \underline{c}) \subset \mathcal{F}(c)$, where if $\alpha = (\infty, \ldots, \infty)$ then $\mathcal{F}(c) = \{\infty\}$, otherwise, $\mathcal{F}(c) = \cup_{\alpha_i \neq \infty}\{x \in \mathbb{R}_+^n | x_i = \alpha_i\}$.*

Proof. Fringe$(x, \underline{c}) = x + \min\{\alpha_i - x_i \mid 1 \leq i \leq n\} \times \mathbf{1}$. Assume that $\min\{\alpha_i - x_i \mid 1 \leq i \leq n\} = \alpha_j - x_j$. If $\alpha_j - x_j = \infty$, then for each i, $\alpha_i = \infty$. As a result, Fringe$(x, \underline{c}) = \{\infty\}$. If $\alpha_j - x_j < \infty$, then x_j, the j-th coordinate of the Fringe$(x, \underline{c})$, is $\alpha_j = x_j + (\alpha_j - x_j)$. $\;///$

Assume that $c \in c(\mathcal{X})$ and $x \in \underline{c}$. In a layman language, Fringe $(x, \underline{c})$ is the final point that an imaginary person can arrive at, if he/she starts from the point x and moves on a line in the direction of the vector $\mathbf{1} = (1, \ldots, 1)$, while his/her trajectory of movement avoids violating c. In a similar way, moving in the direction of $(-1, \ldots, -1)$ can be considered.

Definition 3. *Assume that $c \in \mathbf{c}(\mathcal{X})$ and $x \in \underline{c}$, we shall write* [6] $\Delta(x, \underline{c}) = \sup\{t \geq 0 \mid x - t \times \mathbf{1} \in \underline{c}$, *where* $\mathbf{1} = (1, \ldots, 1)$. *Let us write AntiFringe*$(x, \underline{c}) = x - \Delta(x, \underline{c}) \times \mathbf{1}$. *Moreover, if* $c_1 \in \mathbf{c}(\mathcal{X})$ *and* $\underline{c_1} \subset \underline{c}$ *then define AntiFringe*$(\underline{c_1}, \underline{c}) = \cup_{x \in \underline{c_1}}$*AntiFringe*$(x, \underline{c})$.

The following result, which is also depicted in Fig. 6 explains that to calculate the Fringe, we can use the AntiFringe.

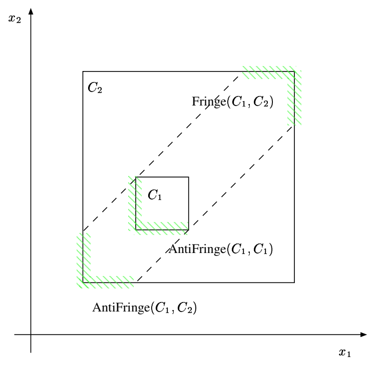

Fig. 6. Fringe(C_1, C_2) = Fringe(AntiFringe$(C_1, C_2), C_2)$

Lemma 5. *Assume that* $c_1 \subset c_2$ *are both in* $\mathbf{c}(\mathcal{X})$ *then*

$$\textit{Fringe}(\underline{c_1}, \underline{c_2}) = \textit{Fringe(AntiFringe}(\underline{c_1}, \underline{c_1}), \underline{c_2}).$$

Proof. The proof is straight forward and omitted.

The following lemma is the equivalent of Lemma 4 for AntiFringe.

Lemma 6. *Assume that* $c \in \mathbf{c}(\mathcal{X})$ *and* $\beta = (\beta_1, \ldots, \beta_n)$ *is the Bottom corner point of the region created by discarding all atomic formulae of the from* $x_i - x_j \sim q$. *Then AntiFringe*$(\underline{c}, \underline{c}) = \bigcup_{i=1}^{n} \underline{P_i(c)}$ *where for* $1 \leq i \leq n$, $P_i(c)$ *is created from c by*

1. *replacing* $x_i > q$ *or* $x_i \geq q$ *with* $x_i = \beta_i$, *and*
2. *for* $j \neq i$ *replacing any* $x_j > q$ *with* $x_j \geq q$.

Proof. Assume that $z \in$ AntiFringe $(\underline{c}, \underline{c})$, then there is $y \in \underline{c}$ such that $z = y - \Delta(y, \underline{c}) \times \mathbf{1}$. Using a similar discussion to Lemma 2, we can prove that AntiFringe $(y, \underline{c}) = y - \min\{y_i - \beta_i \mid 1 \leq i \leq n\}$. Hence, if $\min\{y_i - \beta_i \mid 1 \leq i \leq n\} = y_r - \beta_r$, we show that $z \in \underline{P_r(c)}$. There are four types of atomic formulae in $\underline{P_r(c)}$.

1. Atomic formulae of c of the form $x_i > q$ or $x_i \geq q$, which are replaced with $x_i = \beta_i$.

[6] Notice, the supremum always exists.

2. For $j \neq i$, atomic formulae of the form $x_j > q$, which are replaced with $x_j \geq q$.
3. For $1 \leq j \leq n$, atomic formulae of c of the form $x_j < q$ or $x_j \leq q$.
4. For $1 \leq j, i \leq n$ and $i \neq j$, atomic formulae of c of the form $x_j - x_i \sim q$.

We shall prove that z satisfies formulas which are created from the above atomic formulas. Clearly, z satisfies formulae of type 1 above, since $z_r = y_r - \varDelta(y, \underline{c}) = y_r - (y_r + \beta_r) = \beta_r$.

By the definition of $\varDelta(y, \underline{c})$ there is an increasing sequence $\{t_k\}$ such that $\lim_k t_k = \varDelta(y, \underline{c})$. Hence $z_k = y - t_k \times \mathbf{1} \to y - \varDelta(y, \underline{c}) = z$ and for each k, $y - t_k \times \mathbf{1} \in \underline{c}$. Using this sequence, we can show that z satisfies atomic formulae of type 2–4. For example, if c has an atomic formula of the form $x_j \geq q$ for $j \neq i$ then since $y - t_k \times \mathbf{1} \in c$, $y_i - t_k > q$. As a result $z_i \geq q$.

The other two conditions can be proved similarly. Conversely, assume that $z \in \underline{P_r(\underline{c})}$. There are two cases.

Case 1: $z \in \underline{c}$. In this case, we claim $\varDelta(z, \underline{c}) = \sup\{t \geq 0 \mid z - t\mathbf{1} \in \underline{c}\} = 0$. Otherwise $\varDelta(z, \underline{c}) > 0$. Then $\exists t > 0, z - t\mathbf{1} \in \underline{c}$. As a result, $z_r - t = \beta_r$. Since $z \in \underline{c}$, $z_r = \beta_r$. Hence, $t = 0$, which is a contradiction. Consequently, $z =$AntiFringe$(z, \underline{c}) \in$ AntiFringe$(\underline{c}, \underline{c})$.

Case 2: $z \notin \underline{c}$. In this case, since $z \in \overline{(\underline{c})}$, the topological closure. Notice, Topological closure, by Lemma 1 is created by replacing all $>, <$ with $\geq, \leq$, respectively. Now, using the definition of closure, there is a point y on the half line $\{y + t \times \mathbf{1} \mid t > 0\}$ which also belongs to $\underline{c}$, then $z =$AntiFringe$(y, \underline{c})$. ///

The next result presents a method of detecting the TAL via Presburger Sentences.

Theorem 2. *Assume that* $c_1, c_2 \in \mathbf{c}(\mathcal{X})$ *and* $\underline{c_1} \subset \underline{c_2}$. *Let* c_3 *denote disjunction of finite number of elements of* $\mathbf{c}(\mathcal{X})$. *Then the following is valid.*

$$\mathit{Fringe}(\underline{c_1}, \underline{c_2}) \subset \underline{c_3}. \tag{2}$$

if and only if for each $1 \leq i \leq n$

$$\forall x \in \underline{P_i(c_1)}((\exists t \geq 0\ x + t\mathbf{1} \in \mathcal{F}(c_2'))) \Rightarrow x + t\mathbf{1} \in \underline{c_3}.) \tag{3}$$

where $\underline{P_i(c_1)}$ *is defined in the Lemma 6,* c_2' *is the rectangular region created from c by cancelling atomic formulae of the form* $x_i - x_j \sim q$ *and* $\mathcal{F}(c_2')$ *is defined in Lemma 4.*

Proof. Direct result of applying Lemma 2, 3, 4 and Lemma 6.

The equation (3) above is an RPS. $P_i(c_1)$, $\mathcal{F}(c_2'))$ and c_3 are first order logic formulae on the atomic formulae $x_i \sim q$ and $x_i - y_i \sim q$, where q is a rational number. Moreover, the formula is closed, as variables $x_1, \ldots, x_n$ and t are in the scope of $\forall x$ and $\exists t$.

7 Time Action Lock Checker (TALC)

We have developed a tool called *Time Action Lock Checker* (TALC), which works in conjunction with UPPAAL version 3.2.X and runs under Linux. Fig. 7 depicts the architecture of TALC, which consists of the following five components [26]:

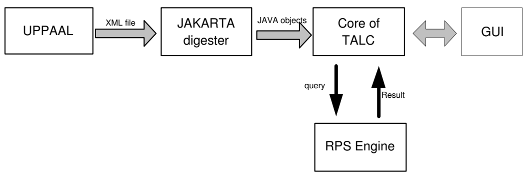

Fig. 7. The Architecture of TALC

– Model Checker UPPAAL, which saves network of Timed Automata models as eXtensible Markup Language (XML)[12] files.
– Jakarta Digester [18] transfers XML files to Java objects which captures the information regarding the network of Timed Automata in Java.
– Core of TALC implements the theory described in previous section and uses Java objects created by the above component to create a set of Rational Presburger Sentences, which are used to evaluate Time Action Lock freeness of the system, see Theorem 2.
– RPS Engine is a component software based on [24] that evaluates Rational Presburger Sentences. It can be invoked by the Core of TALC and receives Rational Presburger Sentences create by Core of TALC in form of *queries*, evaluates the correctness of the Rational Presburger Sentences and returns the *results* of the evaluation.
– TALC includes a user interface component (GUI) which enables the user to interact with the system.

The next example applies the TALC to the verification of a simplified media Synchronisation Protocol motivated by an example studied in [17].

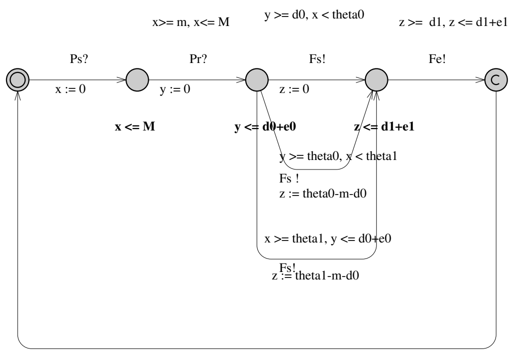

Fig. 8. Synchronised Protocol Module

Example: Fig. 8 depicts a Timed Automata model of Synchronisation Protocol Module (SPM), used in a video streaming system for reducing the jitter caused by the network delay. The Timed Automaton includes signals *Ps* and *Pr*, corresponding to *sending* and *receiving* of packets, respectively. The outputs of SPM are signals *Fs* and *Fe*, which mark *starting* and *ending* of the *frame* display, respectively.

There is a clock x, which resets when a packet is sent *i.e.*, *Ps*?. For the signal *Pr* to receive, there is a delay with a value in $[m, M]$, where m and M are constant rational numbers. On the arrival of *Pr* another clock y resets. At this stage packets are decoded into frames. Decoding requires a delay in $[d_0, d_0 + e_0]$, where d_0 and e_0 are constant rational numbers. According to the time of the sending of a packet, measured by x, and the time of arrival of that packet, measured by y, there are three possible scenarios. Each scenario compares the value of x and y with constants θ_0 and θ_1 and assigns the value of a new clock z, which will be used to determine the time of termination of the display of frames marked by the output signal *Fe*. Using TALC, we can test the above Timed Automaton and infer that the system is deadlock free.

8 Conclusion

This paper presents a geometric method for the detection of Time Action Lock in Timed Automata, the paper makes use of the geometry of $\mathbb{R}^n$ to identify the part of the specification that may result in Time Action Lock. The emphasis is on testing and identifying possible sources of defects in the design by studying various regions representing guards and invariants of the Timed Automata. In particular, the method is based on the study of subsets of such regions known as Fringes. Evaluating various first order closed formulae are carried out via a Real Presburger Sentences solver. The method is implemented in a tool called TALC and is successfully applied to the verification of a simple communication protocol.

References

1. R. Alur and D.L. Dill: *A Theory for Timed Automata*, In Theoretical Computer Science **125**, pp.183–235, 1994.
2. T. Amnell, G. Behrmann, J. Bengtsson, P. R. D'Argenio, A. David, A. Fehnker, T. Hune, B. Jeannet, K. G. Larsen, O. Möller, P. Pettersson, C. Weise and W. Yi: *UPPAAL—Now, Next and Future* In proceedings of Modelling and Verification of Parallel Processes (MOVEP2k), LNCS **2067**, pp.100–125, 2001.
3. T. Amon, G. Borriello, T. Hu and J. Liu *Symbolic Timing Verification of Timing Diagrams Using Presburger Formulas* 34th ACM/IEEE Design Automation Conference (DAC), June 1997.
4. C. W. Barett, D. L. Dill, and J. R. Levitt: *Validity Checking for Combinations of Theories with Equality,* LNCS **818**, pp.187–201, 1996.
5. J. Bengtsson, W. O. D. Griffioen, K.J. Kristoffersen, K. G. Larsen, F. Larsson, P. Pettersson and W. Yi: *Verification of an Audio Protocol with Bus Collision Using UPPAAL*, In Proceedings of the 8th International Conference on Computer-Aided Verification, LNCS **1102**, pp.244–256, 1996.

6. J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson and W. Yi: *UPPAAL, a Tool suite for automatic verification of Real-time systems* In Proceedings of Workshop on Hybrid Systems III: Verification and Control, LNCS **1066** pp.232–243, 1995.
7. S. Bornot and J. Sifakis *On the composition of hybrid systems* In Hybrid systems: computation and Control, LNCS **1386**, pp.49–63, 1998.
8. H. Bowman *Time and action lock freedom properties of timed automata*, In M. Kim, B. Chin, S. Kang, and D. Lee, editors, Formal Techniques for Networked and Distributed Systems, pp.119–134. Kluwer Academic Publishers, 2001.
9. H. Bowman, G. Faconti, and M. Massink: *Specification and verification of media constraints using UPPAAL*, In Proceedings of Design, Specification and Verification of Interactive Systems '98, Markopoulos and P. Johnos, editors, pp.261–277 Springer, 1998.
10. T. Bultan, R. Gerber, W. Pugh *Symbolic Model Checking of Infinite State Programs Using Presburger Arithmetic* Proceedings of the 9th International Conference on Computer Aided Verification (CAV '97), Orna Grumberg, ed., LNCS **1254**, pp.400–411, 1997.
11. N. Bourbaki: *Elements of Mathematics —General Topology—*, chapters 1-4, Springer-Verlag 1980.
12. *eXtensible Markup Language*(XML), http://www.w3.org/XML/.
13. J. Ferrante and C. Rackoff: *A Deicision Procedure for the first Order Theory of Real Addition with order,* SIAM Journal of Computation **4**, pp.69–76, 1975.
14. A. Finkel and J. Leroux *How to compose Presburger-accelerations: Applications to broadcast protocols* In Proc. 22nd Conf. Found. of Software Technology and Theor. Comp. Sci. (FST and TCS 2002), Kanpur, India LNCS **2556**, pp.145–156, 2002.
15. K. Havelund, A. Skou, K. G. Larsen and K. Lund: *Formal Modelling and Analysis of an Audio/Video Protocol: An Industrial Case Study Using UPPAAL* In Proceedings of the 18th IEEE Real-Time Systems Symposium, pp.2–13, 1997.
16. T. A. Henzinger *Sooner is Safer Than Later*, Inf. Process. Lett. 43(3) pp. 135-141, 1992.
17. T. Higashino, A. Nakata, K. Taniguchi, A. R. Cavalli: *Generating Test Cases for a Timed I/O Automaton Model*, Proceedings of the Twelfth IFIP Workshop on Testing of Communicating Systems (IWTCS'99), pp.197–214, 1999.
18. Apache Jakarta Project, Common digester, (jakarta.apache.org/commons/digester/)
19. G. Kreisel and J. Krivine *Elements of Mathematical Logic; Model Theory* North-Holland Publishing Company, 1967.
20. H. Lönn and P. Pettersson: *Formal Verification of a TDMA Protocol Start-Up Mechanism*, In Proceedings of 1997 IEEE Pacific Rim International Symposium on Fault-Tolerant Systems, pp.235–242, 1997.
21. K. G. Larsen, Paul Pettersson and W. Yi: *UPPAAL in a Nutshell*, In Springer International Journal of Software Tools for Technology Transfer **1(1+2)** 1997.
22. R. Milner, *Communication and concurrency*, Prentice Hall, Upper Saddle River, NJ, 1989.
23. *The Omega Project,* `http://www.cs.umd.edu/projects/omega/`
24. N. Shibata, K. Okano, T. Higashino and K. Taniguchi: *A decision algorithm for prenex normal form rational Presburger sentences based on combinatorial geometry,* Proceedings of the 2nd International Conf. on Discrete Mathematics and Theoretical Computer Science and the 5th Australasian Theory Symposium (DMTCS'99+CATS'99), pp.344–359 1999.
25. T. R. Shiple, J. H. Kukula, and R. K. Ranjan: *A Comparison of Presburger Engines for EFSM Reachability,* LNCS **1427**, p.280, 1998.
26. D. Gruntz, S. Murer and C. Szyperski *Component Software - Beyond Object-Oriented Programming*, Second Edition, Addison-Wesley, 2002
27. S. Tripakis, *Verifying Progress in Timed Systems*, In ARTS'99,Formal Method for Real-Time and Probabilistic Systems Bamberg, LNCS **1601**, 1999.
28. F. Wang, G. Hwang and F. Yu *TCTL Inevitability Analysis of Dense-Time Systems* LNCS **2759**, pp. 176–187, 2003.