On Checking Timed Automata for Linear Duration Invariants[*]

Victor Adrian Braberman
Department de Computacion
FCEyN-UBA, Argentina
vbraber@dc.uba.ar

Dang Van Hung[†]
The United Nations University
UNU/IIST, P.O.Box 3058 Macau
dvh@iist.unu.edu

Abstract

In this work, we address the problem of verifying a Timed Automaton for a real-time property written in Duration Calculus in the form of Linear Duration Invariants. We present a conservative method for solving the problem using the linear programming techniques. First, we provide a procedure to translate Timed Automata to a sort of regular expressions for timed languages. Then, we extend the linear programming-based approaches in [8] to this algebraic notations for the timed automata. Our results in this paper are more general than the ones presented in [8]. Namely, Timed Automata are our starting point, and we can provide an accurate answer to the problem for a larger class of them.

1 Introduction

Timed Automata [2] (TA) are one of the most widely used formalisms to model real-time systems. Linear Duration Invariants (LDI) are a fragment of Duration Calculus [6] to express the linear constraints on the accumulated time for the presence of system states over the behaviour of real-time systems. They are first presented in [15] where it is shown that the reachability problem for the Integration Graphs, a class of Hybrid Automata, can be reduced to the verification for a LDI of the automata.

Our goal is to develop a technique to verify algorithmically for this kind of real-time properties of the real-time systems modelled by Timed Automata. To our knowledge, the existing theoretical methods which works for the whole class of TA seem to be useless even for a small example because of the very high complexity.

In [15] the authors present an algorithm for solving the problem which based on two techniques: digitisation and mixed linear integer programming. Digitisation is a way to obtain a discrete time automaton which generates the integer runs of the original dense time version. Naive digitisation is based on the region graph construction which produces huge graphs depending on the size of the constants involved in comparisons [2]. Besides, the mixed linear integer programming posses a very high complexity. In [1] an algorithm to compute accumulated delays is given. It is even more complex than the one presented in [15] and do not work for negative coefficients for durations in the LDI. In [9] a general approach to model-checking in discrete Duration Calculus is presented. This approach is based on inclusion of regular languages. Obtaining regular languages through the digitisation of the TA as well as the transformation of the Duration Calculus formula into finite state automata are the main sources of the high complexity.

In order to get more practical algorithms there are several proposals based on linear programming techniques [7, 8, 12]. The common idea is to represent a real time system as a language described by some sort of timed regular expression. These works are based on the fact that if the expression to verify for a LDI is finite, i.e. there is no repetition in the expression, the verification is done by a linear programming procedure, maximising the body of the LDI subject to the timing constraints that gives the linear bounds on the size of variables which represent the duration of location traversals. Then, most of the research effort is devoted to searching for a way to reduce the infinite case to the finite one, i.e. to eliminate repetition. The strength of these techniques is the reuse of well studied and efficient set of linear programming tools avoiding digitisation. Their weakness lies on the fact that the starting points are those regular expressions which can only represent a small class of TA.

In this paper, we achieve our goal by generalising the ideas of these methods for TA. In order to cope with them, we give an algebraic formalism for expressing the behaviour of the whole class of timed automata along with a translation procedure. Based on the translation results, we provide conservative analysis (“yes” , “no”, “don’t know” answers) to the problems using linear programming tech-

[*]The paper was written during the fellowship of the first author with The United Nations University, International Institute for Software Technology from September 1997 to March 1998

[†]On leave from Institute of Information Technology, Hanoi, Vietnam

niques. In comparison to the methods in [7, 8], our method can work directly on the whole class of TA and give the accurate answer to the problem for a subclass larger than any previously defined in these approaches.

The paper is organised as follows. In the next section, we recall some basic notions on TA and LDI. A new algebraic formalism for expressing the behaviour of TA will be given in Section 3 which will be the base to explain the technique presented in Section 4. The last section is the conclusion of the paper.

2 Basic Notations

First, we introduce some basic notations for sequences that will be used in the sequel.

Let s be a sequence. Then, $|s|$ will denote the length (i.e. number of elements) and s_i $(0 \leq i \leq |s|-1)$ will denote the ith element of the sequence s. For $0 \leq i \leq j \leq |s|-1$, let $s_{i]}$ denote the prefix of s that ends with the ith element, $s_{[i}$ the suffix of s that starts from the ith element and $s_{[i,j]}$ the subsequence from the ith element to the jth element inclusively. If the sequence s is not empty, its last element (i.e. $s_{|s|-1}$) will be denoted by $last(s)$. The concatenation of two sequences s and s' will be denoted by ss', and a sequence with single element will be identified with its element. Given a set E, given a subset T of E and a sequence s over E. $T \cap s$ will denote the intersection between T and the underlying set of s. Let $last\_T\_in\_s$ be the natural number k such that $s_k \in T$ and $\forall l,\ k < l < |s|$ it holds that $s_l \notin T$.

In this paper, let $\mathbb{Q}$ denotes the set of rationales and $\Re_{\geq 0}$ the set of non negative reals.

2.1 Timed automata

Definition 1 (Timed Automata) *A timed automaton is a tuple $A = (Q, C, L, \Omega, S, Inv)$ where Q is a finite set of locations, C is finite set of clock variables, L is a finite set of labels, Ω is a set of edges (see bellow), $S \subseteq Q$ is a set of initial locations and Inv is a function from the set of locations to conjunctions of tests of the form $x \leq c$ (the invariant of a location), where $c \in \mathbb{Q}$ and x is a clock variable. An edge is a tuple (q, ϕ, ρ, l, q') where $q, q' \in Q$, $\rho \subseteq C$, and l is a label, ϕ (guard) is a conjunction of tests of the form $x \sim c$, where x is a clock variable, $\sim \in \{\leq, \geq\}$ ($\leq$ for deadline and $\geq$ for delay test) and $c \in \mathbb{Q}$. There is at most one edge for each pair of locations.*

Normally systems are specified as nets of time automata running in parallel which are synchronised at the transitions having the same label.

Definition 2 (Parallel composition) *Let $A_1 = (Q_1, C_1, L_1, \Omega_1, S_1, Inv_1)$ and $A_2 = (Q_2, C_2, L_2, \Omega_2, S_2, Inv_2)$ be TA where $C_1 \cap C_2 = \emptyset$. The parallel composition of A_1 and A_2, denoted by $A_1 || A_2$, is the timed automaton $A = (Q_1 \times Q_2, C_1 \cup C_2, L_1 \cup L_2, \Omega_{1\times 2}, S_1 \times S_2, Inv_{1\times 2})$, where $\Omega_{1\times 2}$ and $Inv_{1\times 2}$ are defined as: For any $(q_1, \phi_1, \rho_1, l_1, q_1') \in \Omega_1, (q_2, \phi_2, \rho_2, l_2, q_2') \in \Omega_2$*

- *if $l_1 = l_2$, then $\Omega_{1\times 2}$ includes $((q_1, q_2), \phi_1 \wedge \phi_2, \rho_1 \cup \rho_2, l_1, (q_1', q_2'))$*
- *if l_1 is not in $L_1 \cap L_2$, then $\Omega_{1\times 2}$ includes $((q_1, q_2), \phi_1, \rho_1, l_1, (q_1', q_2))$*
- *if l_2 is not in $L_1 \cap L_2$, then $\Omega_{1\times 2}$ includes $((q_1, q_2), \phi_2, \rho_2, l_2, (q_1, q_2'))$,*

and for any $(q_1, q_2) \in Q_1 \times Q_2$ $Inv_{1\times 2}((q_1, q_2)) = Inv_1(q_1) \wedge Inv_2(q_2)$.

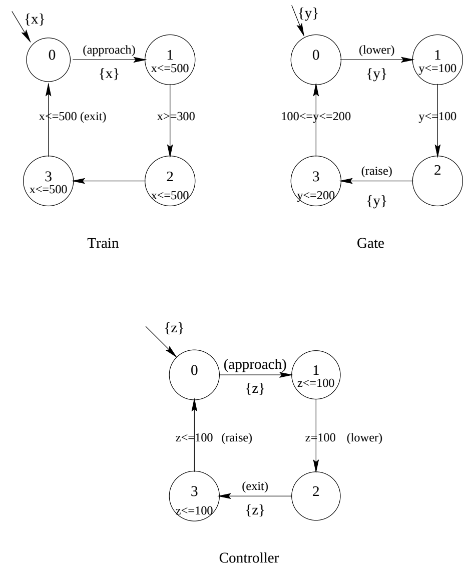

Figure 1. The Railroad Crossing System

Example 1 *The figure 1 shows the three components of the Railroad Crossing System as presented in [14]. They are synchronised at the labels* **approach, exit, lower, down**. *When a train approaches the crossing, it sends a signal* **approach** *to the Controller and enters the crossing at least* 300 *time units later. When leaving the crossing it sends a signal* **exit** *to the Controller within* 500 *time units after the signal* **approach** *has been sent. The Controller sends a signal* **lower** *to the gate exactly* 100 *time units after it has received the signal* **approach**, *and sends a signal* **raise** *within* 100 *time units after it has received the signal* **exit**. *The gate responds to the signal* **lower** *by moving down within* 100 *time units, and responds to the signal* **raise** *by moving up between* 100 *and* 200 *time units.*

Now we define finite timed language to express the behaviour of timed automata.

Definition 3 (Timed word) *A Timed word over an alphabet E is a pair (σ, τ), where σ is an finite sequence of elements of E, and τ is an finite monotonically increasing sequence of $\Re_{\geq 0}$ and both have the same length. A set of timed words over E is called a timed language over E.*

The set of *transitions* over an alphabet Q, denoted by T_Q, is defined as the set $(Q \cup \{\bot\}) \times Q$. The symbol $\bot$ will be used for defining starting transitions of automata. Let π_1, π_2 denote the projections over the first and the second component of transitions respectively.

Let A be a timed automaton as in Definition 1. A *clock valuation* is a function $v : C \to \Re_{\geq 0}$. In this paper we also consider clock valuations as clock-indexed vectors. For a clock valuation v and a set $\rho \subseteq C$ let $Reset_\rho(v)$ denote the valuation defined by

$$Reset_\rho(v)(c) = \begin{cases} 0 & \text{if } c \in \rho, \\ v(c) & \text{otherwise.} \end{cases}$$

We use $\mathbf{1}$ to denote the unit vector $(1, \ldots, 1)$, and $\mathbf{0}$ to denote the 0-vector $(0, \ldots, 0)$ of dimension $|C|$.

A *state* of the timed automaton A is a pair $(q, v) \in Q \times \hbar$ for which $Inv(q)$ holds on v.

An *run* of the automaton A is a sequence

$$r = (q_0, v_0) \overset{\omega_1, \tau_1}{\to} (q_1, v_1) \overset{\omega_2, \tau_2}{\to} \ldots \overset{\omega_n, \tau_n}{\to} (q_n, v_n)$$

where (q_i, v_i) are states of A, $\omega_i \in \Omega$, $\tau_i \in \Re_{\geq 0}$ satisfying

Initialisation: $q_0 \in S$; $v_0 = \mathbf{0}$

Monotonicity: $0 \leq \tau_i \leq \tau_{i+1}$ for all natural numbers $i < n$

Succession: $\omega_i = (q_i, \phi_i, \rho_i, l_i, q_{i+1})$, where the condition $\phi_i(v_{i-1} + (\tau_i - \tau_{i-1})\mathbf{1})$ holds, $v_i = Reset_\rho(v_{i-1} + (\tau_i - \tau_{i-1})\mathbf{1})$, and the condition $Inv(q)(v_{i-1} + t\mathbf{1})$ holds for all $0 \leq t \leq \tau_i - \tau_{i-1}$

The timed word over the alphabet T_Q accepted by the run r above is defined as

$$((\bot, q_0), (q_0, q_1), ..., (q_{n-1}, q_n), 0\tau_1 ... \tau_n) .$$

The timed language $L(A)$ of the timed automaton A is the set of finite timed words over T_Q that has an accepting run. We will only consider in this paper *Non-zeno* Automata (see. [10]), i.e. given any real number t, any finite run of can be extended to a run with the time $\tau_n > t$ (time can progress).

2.2 Linear Duration Invariants

In order to model a real-time system, the automaton A is usually associated with a mapping $\lambda : Q \to 2^P$ which assigns a set of propositional letters to each location which should be interpreted as *true* when the automaton stays at the location.

Example 2 *For the composition automaton of the previous example (whose locations are triples indicating local locations of its three components), let $P = \{Up, Down, MovingUp, MovingDown\}$ and λ be*

$$\lambda((q_1, q_2, q_3)) = \begin{cases} \{Up\} & \textit{if } q_2\textit{=0} \\ \{MovingDown\} & \textit{if } q_2\textit{=1} \\ \{Down\} & \textit{if } q_2\textit{=2} \\ \{MovingUp\} & \textit{if } q_2\textit{=3} \end{cases}$$

Linear Duration Invariants are a particular family of real-time properties that predicates over the runs of timed automata. A LDI is a Duration Calculus formula of the form

$$\Pi \overset{def}{=} l \leq \int 1 \leq u \Rightarrow \textstyle\sum_{b \in B} c_b \int b \leq M ,$$

where B is a finite set of boolean expressions over P, l,c_b and M are reals, and u is either a real or ∞. Given a run of the automaton A, the expression $\int b$ stands for the accumulated time in the run where the expression b evaluates to $true$, i.e. the time that the automaton stays in a location s for which $\lambda(s) \Rightarrow b$. The LDI Π is satisfied by the automaton A when $\sum_{b \in B} c_b \int b$ is less than or equal to M for those runs of A whose 'time length' (τ_n) lies between l and u. Formally,

$$\begin{array}{lll} A \models_\lambda \Pi & \overset{def}{=} & \forall (\sigma, \tau) \in L(A) : \\ & & l \leq last(\tau) \leq u \Rightarrow f_{\Pi,\lambda}(\sigma, \tau) \leq M , \end{array}$$

where

$$\begin{array}{lll} f_{\Pi,\lambda}(\sigma, \tau) & \overset{def}{=} & \sum_{1 \leq i < |\sigma|} \alpha(\pi_1(\sigma_i))(\tau_i - \tau_{i-1}) \\ \alpha(q) & \overset{def}{=} & \sum_{\{b \in B \ | \ \lambda(q) \Rightarrow b\}} c_b \end{array}$$

(α assigns to each location the contribution it makes to the linear duration expression.)

Example 3 *Let*

$$\begin{array}{lll} \Pi & \overset{def}{=} & 0 \leq \int 1 < \infty \Rightarrow \\ & & -3 \int (Down \lor Up) + 2 \\ & & \int (MovingDown \lor MovingUp) \leq 1000 . \end{array}$$

For the composition automaton of Example 1 the mapping α associated to the mapping λ is

$$\alpha((q_1, q_2, q_3)) = \begin{cases} -3 & \textit{if } q_2\textit{=0} \\ 2 & \textit{if } q_2\textit{=1} \\ -3 & \textit{if } q_2\textit{=2} \\ 2 & \textit{if } q_2\textit{=3} \end{cases}$$

Let $\sigma = (\bot, 000), (000, 101), (101, 112), (112, 122)$*;* $\tau = 0, 600, 700, 850)$. *Then* $f_{\Pi,\lambda}(\sigma, \tau) = -3 \times 600 + -2 \times 100 + 3 \times 150$.

Since LDIs are universal properties, it is obvious that:

Lemma 1 *For any automata* A *and* A' $L(A) \subseteq L(A') \Rightarrow A' \models_\lambda \Pi \Rightarrow A \models_\lambda \Pi$.

2.3 Simplification of the Problem

Now we make a simple conversion to the problem to get rid of time bounds in the LDI property. Suppose that we want to verify a LDI over all those runs A of the length between l and u. Define a new automaton A' from A by making the following changes to A:

- Add a new clock, namely z,
- Add to all invariants the condition $z \leq u$,
- Add a new "trap" location, namely "final". Associate the invariant *True* to this location and add to it a self loop with the *true* condition and the empty set of reset clock variables (stuttering step).
- Add edges from all original locations to the trap location with the condition $l \leq z \leq u$ and the empty set of reset clock variables.
- Extend λ by assigning any value to "final" (lets call this extension λ').

It is easy to see that the obtained automaton A' has the property that its runs ended in the trap location *final* are exactly the runs that are relevant to the LDI satisfaction. That is,

$$\begin{aligned} A \models_\lambda \Pi \quad \iff \quad & \forall(\sigma, \tau) \in L(A') : \\ & ((\pi_1(last(\sigma)) \neq \text{“}final'' \wedge \\ & \pi_2(last(\sigma)) = \text{“}final'') \\ & \Rightarrow f_{\Pi,\lambda'}(\sigma, \tau) \leq M) \end{aligned}$$

It is obvious that if the original automaton A is non-zeno then so is the automaton A'. Moreover, every run of the automaton A' is a prefix of a run leading to the trap location.

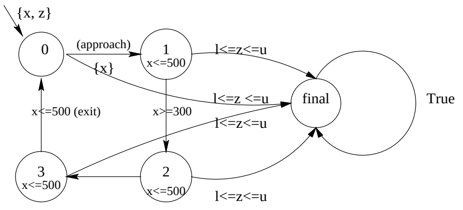

Figure 2. Adding a Trap Location

3 Time Constrained Regular Expressions

In this section we give a descriptive and algebraic representation of the behaviour of TA which will be called Timed Constrained Regular Expressions (TC-RE). TC-RE provides us the necessary insight to formulate the principles of our model checking algorithm proposed later in this paper.

Definition 4 *A TC-RE over an alphabet* Q *is a tuple* $M = (R, \Delta)$ *where* R *is a regular expression over the alphabet* T_Q *and* Δ *is a finite set of triples of the form* $(T, t, \sim c)$*, where* $T \subseteq T_Q$ *(a finite set of transitions over* Q*),* $t \in T_Q$*,* $\sim \in \{\leq, \geq\}$ *and* $c \in \mathbb{Q}$*.*

The intuition behind this definition is that the regular expression R [11] gives the potential untimed sequences of transitions, while Δ establishes a set of constraints for the distance between transition occurrences. In terms of clock metaphor a tuple $(T, t, \sim c)$ in Δ is the analogous of a clock test associated to the transition t (that appears as the second component of the tuple). Then, the first component, T, is viewed as the set of transitions which reset that clock. Roughly speaking, $\sim c$ is a time constraint on the distance between t and the closest previous T-transition.

Definition 5 *Given a TC-RE* (R, Δ) *over* Q*. The language* $L((R, \Delta))$ *represented by* (R, Δ) *is the set of timed words* (σ, τ) *over* T_Q *satisfying:*

- $\sigma \in R$ *(*σ *is a transition sequence described by the regular expression),*
- $\tau_0 = 0$ *(initialisation),*
- $\forall 0 \leq i < |\sigma|$:
  $((T, \sigma_i, \sim c) \in \Delta \wedge (T \cap \sigma_{i-1]} \neq \emptyset$
  $\Rightarrow \tau_i - \tau_{last\_T\_in\_\sigma_{i-1]}} \sim c))$
  (i.e. τ *satisfies the time constraints on the distance between event instances).*

Hereafter we write $\tau \in Sol(\sigma, \Delta)$ to express that (σ, τ) is a timed word satisfying the conditions of the last two items of Definition 5.

Example 4 *Let R*= $(\bot, 0)\ ((0, 1)(1, 2)(2, 3)(3, 0))^*$,

$$\Delta = \left\{ \begin{array}{c} (\{(\bot, 0), (0, 1)\}, (1, 2), \geq 300), \\ (\{(\bot, 0), (0, 1)\}, (1, 2), \leq 500), \\ (\{(\bot, 0), (0, 1)\}, (2, 3), \leq 500), \\ (\{(\bot, 0), (0, 1)\}, (3, 0), \geq 300) \end{array} \right\}.$$

Then, (R, Δ) *is a TC-RE that represents all the timed words leading to the location* 0 *of the automaton "Train" in Fig. 1*

The notion of TC-RE leads to a clear separation between the untimed structure and timing constraints. The structured nature of classical Regular Expressions is extremely helpful for developing the principles of our algorithm. The following facts are obvious from the above definition.

Fact 1 *Let R and R' be equivalent REs (i.e. they recognise the same language), and Δ be a set of constraints. Then* $L(R,\Delta) = L(R',\Delta)$.

Fact 2 $L(R, \Delta \cup (T, t, \sim c)) \subseteq L(R, \Delta)$.

4 Principles for Our Model Checking Algorithm

It is straightforward to derive from the timed automaton A' a TC-RE that defines the set of timed words of A' which lead to the final trap location. However, we want a TC-RE with a 'good' structure which allows a simple treatment for the Kleene closure subexpressions in verifying for a LDI. Such a TC-RE is said to be 'well behaved and operational'. The first step of our model-checking procedure is to obtain a 'well behaved and operational' from the input timed automaton. Then, we divide the obtained TC-RE into two cases: finite (with no Kleene closure occurrence) and infinite (with Kleene closure occurrence) ones. In the finite case we show how to apply linear programming techniques to solve our model-checking problem. For the infinite case we present some techniques to reduce the number of Kleene closure subexpressions or to infer that the LDI is violated. In the following, we give the more details of our idea.

4.1 Well Behaved and Operational TC-RE

Now we restrict ourselves to some special class of TC-RE whose properties can simplify our algorithm (for dealing with infinite RE) without restricting the expressive power.

Definition 6 *A sequence σ of transitions is Repeatable (abbreviations: $Repeatable(\sigma)$) iff $\pi_1(\sigma_1) = \pi_2(last(\sigma))$ (recall that a transition is a pair of locations).*

Definition 7 *A repeatable word σ is non transient for Δ (abbreviations: $NTransient_\Delta(\sigma)$) iff $\exists i < |\sigma| : \exists (T, \sigma_i, \geq c) \in \Delta : c > 0 \wedge T \cap \sigma \neq \emptyset$ (i.e. Non-transientness means that if the word is repeated, time does elapse).*

Definition 8 *A TC-RE (R,Δ) is a Well Behaved TC-RE if it satisfies the following properties:*

Fusion Closure: *For any transitions t, t' such that $\pi_2(t) = \pi_2(t')$, for any sequences of transitions σ, σ', θ and θ' it holds $(\sigma t\theta \in R \wedge \sigma' t' \theta' \in R) \Rightarrow (\sigma t \theta' \in R \wedge \sigma' t' \theta \in R)$.*

Feasibility: $\forall \sigma \in R : Sol(\sigma, \Delta) \neq \emptyset$.

Nodeadline-Constrained Non-Zero Iterations:
$\forall \sigma \in R\, \forall k < |\sigma|\, \forall (T, \sigma_k, \leq c) \in \Delta :$
$(T \cap \sigma_{k-1]} = \emptyset \vee$
$(\forall i, j : last\ T\ in\ \sigma_{k-1]} < i < j < k \wedge$
$Repeatable(\sigma_{[i,j]})$
$\Rightarrow (\forall \tau \in Sol(\sigma, \Delta) : \tau_j - \tau_i = 0))$.

Nodelay-Constrained Non-Transient Iterations: $\forall \sigma \in R\, \forall k < |\sigma|\, \forall (T, \sigma_k, \geq c) \in \Delta :$
$(T \cap \sigma_{k-1]} = \emptyset \vee$
$(\forall i, j :\ last\ T\ in\ \sigma_{k-1]} < i < j < k \wedge$
$Repeatable(\sigma_{[i,j]})\ \wedge\ NTransient_\Delta(\sigma_{[i,j]})$
$\Rightarrow \forall \tau \in Sol(R, \Delta) : \tau_{i-1} - \tau_{last\ T\ in\ \sigma_{k-1]}} > c))$.

The first property is natural since symbols are pair of locations. The second property avoids the case in which words described by the RE have no associated solution for the timing constraints imposed by Δ. The last two properties have a more technical nature and simplify the analysis of star subexpressions (kleene closure). Intuitively, the property *No-deadline-Constrained Non-Zero Iterations* rules out the case that deadlines ranging over a repeatable sub word where time elapses. On the other hand, the property *No-delay-Constrained Non-Transient Iterations* ensures that the time distance from a repeatable part to the previous reset is required to be greater than any constant that appears in a delay constraint even before the repeatable sub word occurrence, making its test redundant. Then, roughly speaking, these properties affirm that the number of iterations does not affect, and is not affected by timing constraints. These properties can be achieved for a timed automaton using techniques explained in the next section.

Example 5 *Let R =* $(\perp, 0)(0,1)\ ((1,2)(2,1))^*\ (1,3)$*, and*

$$\Delta = \left\{ \begin{array}{c} (\{(\perp,0)\}, (0,1), \geq 100), \\ (\{(0,1),(2,1)\}, (1,2), \geq 50), \\ (\{(\perp,0)\}, (1,3), \geq 100) \end{array} \right\}.$$

Then, (R,Δ) is a well behaved TC-RE. Note that the iteration is non transient since the runs must remain in the location 1 for at least 50 time units.

Definition 9 *A TC-RC (R,Δ) is operational if it satisfies the condition: for all $\sigma \in R$, $0 \leq i < |\sigma|$, $\tau \in Sol(\sigma_{i]}, \Delta)$ there exists θ sequence of transitions such that $(\sigma_{i]}\theta, \tau') \in L(R,\Delta)$ and $\tau = \tau'_{i]}$*

The operationality means that partial solutions are part of total solutions and it is intimately connected to non-zenoness. Note that the TC-RE in the previous example is also operational since for any compatible time assignment for some prefix of R there is always a way to extend it to reach the location 3.

In the next section we show how to obtain a well behaved and operational TC-RE from a timed automaton that satisfies Non-zenoness. That TC-RE recognises the relevant language of timed words leading to the final location.

4.2 Problem Transformation in Terms of Well Behaved and Operational TC-RE

Given a timed automaton B and a final location *final*, we could apply Kleene procedure to B to obtain a regular expression that recognises all the untimed words of transitions that leads to the location *final* without using the stuttering step at the location final location [11]. Then, Δ is obtained in a simple way as follows.

For each edge, namely (q, ϕ, ρ, l, q') do the following

- For each test $x \sim c \in \rho$ add the tuple $(T, (q, q'), \sim c)$ to Δ, where T is the set of transitions that reset the clock x (remember that this set includes also the initial ones, i.e. $(\perp, s) \in T$ for all $s \in S$).
- For each test $x \leq c \in Inv(q)$ add the tuple $(T, (q, q'), \leq c)$ to Δ, where T is the set of transitions that reset the clock x including the the initial ones as above.

It is easy to see that if the location *final* is a trap location (no transition but a stuttering leaves that location) the obtained TC-RE (denoted by $\Delta Kleene(B)$) recognises exactly the language of timed words of B leading to *final*. Then we have the following observation

Fact 3 *$A \models_\lambda \Pi$ iff $(\forall(\sigma, \tau) \in L(\Delta Kleene(A')) : f_{\Pi,\lambda}(\sigma', \tau) \leq M)$, where A', λ' are the automaton and the mapping obtained by following the procedure of section 2.3.*

Therefore in order to verify for the LDI (described earlier) of the timed automaton A, all we have to do is to check whether $f_{\Pi,\lambda}(\sigma, \tau) \leq M$ is satisfied by all the timed words described by the obtained TC-RE. Is the obtained TC-RE well behaved and operational?

Since symbols are pair of locations the resulting language has the **Fusion Closure** property.

And also, since every run of A' is a prefix of a run leading to the trap state (this property comes from the fact that A is non-zeno), it is easy to see that the obtained TC- RE is operational.

Unfortunately, the last three properties are not guaranteed. In order to obtain a TC-RE satisfying the property feasibility we can apply Kleene conversion to the reachability graph ($RG(A')$) obtained from the automaton A' . The Reachability Graph is a well known concept in many timed formalisms [14, 13, 4].

In particular, the $RG(A)$ can be seen as an automaton that recognises (up to renaming, let us call it β) the same language as A. It is built unfolding symbolically the original graph in such a way that the language recognised by the underlying graph is feasible. This implies immediately the **feasibility** of its associated TC-RE.

Example 6 *The reachability graph for the timed automaton $(train||gate||controller)'$ is shown in Figure 3, where a final location and an extra clock for the observation interval [0,∞) were added.*

It is easy to see that $\Delta Kleene(RG(A'))$ is **operational** as well. It is because that A' is the automaton obtained from a non zeno automaton A using the procedure of Section 2.3. In fact, runs in the $RG(A')$ can be seen as runs of A'. Then, any violation of the operationality of $\Delta Kleene(A')$ would imply zenoness of the original automaton A.

Moreover, the locations of the reachability graph convey timing information about paths leading to them. This leads to the property 'reachability equivalence' of the TC-RE obtained from $RG(A')$, which is formalised below.

Lemma 2 *The TC-RE (R, Δ) obtained from $RG(A')$ satisfies the property* **Reachability Equivalence**, *i.e. for all $\sigma, \sigma' \in Prefixes(R)$ such that $\pi_2(last(\sigma)) = \pi_2(last(\sigma'))$, for all $(T, t, \sim c) \in \Delta$ if $\exists \theta \in R, i, j \in \mathbb{N} : 0 \leq i < j < |\theta| : \pi_2(\theta_i) = \pi_2(last(\sigma)) \wedge \theta_j = t$ (i.e. exists a future comparison) then:*

1. *$Min\{last(\tau) - \tau_{lastTin\sigma} \mid \tau \in Sol(\sigma, \Delta)\} \leq c \Rightarrow Min\{last(\tau) - \tau_{lastTin\sigma} \mid \tau \in Sol(\sigma, \Delta)\} = Min\{last(\tau) - \tau_{lastTin\sigma'} \mid \tau \in Sol(\sigma', \Delta)\}$, and*
2. *$Max\{last(\tau) - \tau_{lastTin\sigma} \mid \tau \in Sol(\sigma, \Delta)\} \leq c \Rightarrow Max\{last(\tau) - \tau_{lastTin\sigma} \mid \tau \in Sol(\sigma, \Delta)\} = Max\{last(\tau) - \tau_{lastTin\sigma'} \mid \tau \in Sol(\sigma', \Delta)\}$*

It can be seen that reachability equivalence implies the last two properties [5].

Example 7 *The TC-RE that recognises all the time words of the RG in Figure 3, which lead to the location* final *is the following:*

$$
\begin{aligned}
R \stackrel{def}{=} \; & ((A^*(a,b)(b,c)\ldots(i,j)((j,c)B^*(c,d)\ldots(i,j))^* \\
& \left(\begin{array}{l}
(j, final)\oplus \\
(j,c)B^*(c, final)\oplus \\
(j,c)B^*(c,d)(d, final)\oplus \\
\ldots\oplus \\
(j,c)B^*(c,d)\ldots(h, final)\oplus \\
(j,c)B^*(c,d)\ldots(h,i)(i, final)\oplus \\
(j,c)B^*(c,d)\ldots(h,a)(a, final)\oplus \\
(j,c)B^*(c,d)\ldots(h,a)(a,b)(b, final)
\end{array}\right)) \\
& \oplus \left(\begin{array}{l}
A^*(a, final)\oplus \\
A^*(a,b)(b, final)\oplus \\
\ldots\oplus \\
A^*(a,b)\ldots(i, final)
\end{array}\right))
\end{aligned}
$$

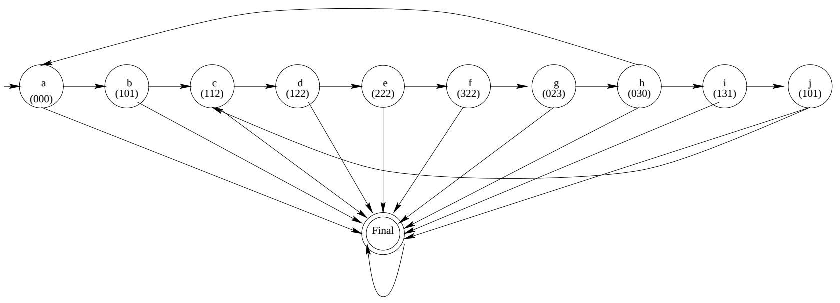

Figure 3. The Reachability Graph

where

$$A = (a,b)\dots(h,a)$$
$$B = (c,d)\dots(j,c).$$

and

$$\Delta = \left\{ \begin{array}{c} (\{(\bot,a),(a,b)\},(b,c),\geq 100), \\ (\{(\bot,a)\},(a,final),\geq 0), \\ (\{(\bot,a),(a,b)\},(b,c),\leq 100), \\ (\{(\bot,a)\},(b,final),\geq 0), \\ (\{(\bot,a),(b,c),(j,c)\},(c,d),\leq 300), \\ (\{(\bot,a)\},(c,final),\geq 0), \\ (\{(\bot,a),(a,b),(h,i)\},(d,e),\leq 500), \\ (\{(\bot,a)\},(d,final),\geq 0), \\ (\{(\bot,a),(a,b),(h,i)\},(d,e),\geq 300), \\ (\{(\bot,a)\},(e,final),\geq 0), \\ (\{(\bot,a),(a,b),(h,i)\},(e,f),\leq 500), \\ (\{(\bot,a)\},(c,final),\geq 0), \\ (\{(\bot,a),(a,b),(h,i)\},(f,g),\leq 300), \\ (\{(\bot,a)\},(f,final),\geq 0), \\ (\{(\bot,a),(f,g)\},(g,h),\leq 100), \\ (\{(\bot,a)\},(g,final),\geq 0), \\ (\{(\bot,a),(g,h)\},(h,i),\leq 200), \\ (\{(\bot,a)\},(h,final),\geq 0), \\ (\{(\bot,a),(g,h)\},(i,j),\geq 100), \\ (\{(\bot,a)\},(final,final),\geq 0), \\ (\{(\bot,a),(g,h)\},(i,j),\leq 100), \\ (\{(\bot,a),(h,i)\},(j,c),\geq 100), \\ (\{(\bot,a),(h,i)\},(j,c),\leq 100), \\ (\{(\bot,a),(g,h)\},(h,a),\geq 100), \\ (\{(\bot,a),(g,h)\},(h,a),\leq 200) \end{array} \right\}$$

Now we formulate the main result of this section.

Lemma 3 $\Delta Kleene(RG(A'))$ *is a well behaved and operational TC-RE for which*

$$A \models_\lambda \Pi \quad \textit{iff} \quad (\forall(\sigma,\tau) \in L(\Delta Kleene(RG(A'))) : f_{\Pi,\lambda'\beta}(\sigma,\tau) \leq M)\,.$$

Therefore, in the rest of the paper we have to deal only with the problem of deciding whether $\forall(\sigma,\tau) \in L(R,\Delta)$: $f_{\Pi,\lambda}(\sigma,\tau) \leq M$ for a well behaved and operational TC-RE (R,Δ).

4.3 Checking Finite TC-RE

A *finite* TC-RE is a TC-RE where first component is a RE having no occurrence of the star (Kleene closure). Finiteness implies that the RE can be rewritten as a finite union of words. The LDI must be satisfied for all the words in which the RE can be decomposed into. Note that this procedure can be done in parallel. Given a word σ and a set of time constraints Δ, we can associate to it a set of variables and a set of linear constraints $C(\sigma,\Delta)$ such that its solutions form the set $Sol(\sigma,\Delta)$. Formally, the set $C(\sigma,\Delta)$ of inequalities on the variable set $(x_i)_{i<|\sigma|}$ is defined by

$$\begin{array}{ll} & \left\{ x_i - x_{last\ T\ in\ \sigma_{i-1]}} \sim c \;\middle|\; \begin{array}{l} i < |\sigma| \wedge \\ (T,\sigma_i,\sim c) \in \Delta \wedge \\ T \cap \sigma_{i-1]} \neq \emptyset \end{array} \right\} \\ \cup & \{x_{i+1} - x_i \geq 0 \mid i+1 < |\sigma|\} \\ \cup & \{x_0 = 0\}\,. \end{array}$$

Then checking the LDI Π under renaming λ over (σ,Δ) is exactly checking whether the maximum of the function $f_{\Pi,\lambda}(\sigma,x)$ subject to $C(\sigma,\Delta)$ is less or equal to M which can be solved by the linear programming techniques. Hereafter we call this maximum $MaxInv(\sigma)$.

Example 8 *Let* σ *be* $(\bot,a)(a,b)(b,c)(c,d)(d,final)$, *and* Δ *be as in the previous example. Then* $C(\sigma,\Delta)$ *is the set of the following inequalities:*

$$\begin{array}{l} 0 = x_0, x_1 - x_0 \geq 0, x_2 - x_1 \geq 0, \\ x_3 - x_2 \geq 0, x_4 - x_3 \geq 0, x_2 - x_1 = 100, \\ x_3 - x_2 \leq 300, x_4 - x_0 \geq 0\,. \end{array}$$

Let Π *be as in Example 3. It is easy to see that* $MaxInv(\sigma)$ *is* 0, *and it is reached at* $(x_0 = 0,\ x_1 = 0,\ x_2 = 100,\ x_3 = 400,\ x_4 = 400)$.

4.4 Eliminating Star Occurrences

In this subsection we present some results aiming at supporting the claim of correctness of our algorithm. These results set the principles to reduce the general case to the finite case. These reductions can be applied to a large class of timed automata.

Let R and P be RE, whenever we say that P is a sub-expression of R we will refer to a particular occurrence of it.

Now we define when a finite set of words (the canonical form of a finite regular expression P) form an 'independent' set. By independency, we intend to mean that if P is the body of a cycle P^* then the timing assignment of each iteration will not depend on the timing assignment of previous one.

Definition 10 *A finite set P of words is independent iff $\forall \sigma, \gamma, \theta \in P : \forall 0 \leq i < |\sigma| : (T, \sigma_i, \sim c) \in \Delta \wedge T \cap \sigma_{i-1]} = \emptyset \wedge T \cap \theta \neq \emptyset \Rightarrow last(\theta) \in T \wedge last(\gamma) \in T$ (i.e. no constraint go from one iteration to another one).*

Roughly speaking, this condition is satisfied when all tests on a clock in the cycle have the corresponding reset appearing before them in the cycle or just before entering the cycle. We present the following theorems for dealing with independent iterations.

Theorem 1 *Let (R, Δ) be a well behaved TC-RE, let P^* be a subexpression of R where P is a finite and independent set of repeatable words satisfying that all the $\gamma \in P$ are non transient and $MaxInv(last(\gamma)\gamma) \leq 0$. Then by replacing P^* in R with $\epsilon \oplus P \oplus PP$, we get a regular expression R' which satisfies: $(R, \Delta) \models_\lambda \Pi$ iff $(R', \Delta) \models_\lambda \Pi$.*

Theorem 2 *Let (R, Δ) be a well behaved and operational TC-RE, let P^* be a subexpression of R where P is finite. Let $\theta \in P^*$ be independent from itself (i.e. the set $\{\theta\}$ is independent). If $MaxInv(last(\theta)\theta) > 0$ then $(R, \Delta) \not\models_\lambda \Pi$*

4.5 Description of the Algorithm

In Section 2.3 we have shown how the original problem can be transformed into the verification of an LDI on a well-behaved and operational TC-RE. That is done by adding a final trap location, performing a symbolic unfold of the automaton (reachability graph) and then applying a simple conversion into TC-RE. Hereafter we sketch an algorithm based on the results presented in the previous sections. For the simplicity of our presentation, we assume that the Automaton has Non-Transient cycles (i.e. time must elapse when the body of a cycle is performed). We believe that this property is not very restrictive in practice.

Given the TC-RE, the algorithm works iteratively with the most inner star occurrences are eliminated by applying Theorem 1 and Theorem 2. If the words which constitute the body of the star are independent then reduction to the finite case can be achieved or counterexample can be found. If we reduce the whole expression to a finite one, we use the techniques presented in section 4.3 to give the answer to the problem.

If the formerly presented strategies do not work, a conservative step is still possible. The idea is basically to make a Kleene-closured finite subexpression independent. This is achieved by eliminating constraints which violate the independence condition (previously, we apply renaming to eliminate just the localised constraints). Note that the former procedure leads to a simpler system which is easier to analyse but with a potential enlargement of the language. Therefore, if after making these manipulations the resulting finite TC-RE satisfies the LDI then we know that the original does satisfy it (see fact 1). If a counterexample is found after a conservative step it must be checked whether it is included into the original language and if this is not the case then the algorithm yields no definitive response ("?").

Example 9 *Observe that cycle $P^* = ((a, b) \ldots (h, a))^*)$ has just a word and generates independent iterations. It is easy to see that $MaxInv(P, \Delta) > 0$. In fact, the automaton can remain at a, d, e, f and g for 0 time units whereas it can remain 100 time units in b, 300 time units in c and 200 time units in h. The value of the objective function is 700 in this case. Then we conclude that the invariant is not satisfied.*

5 Conclusions

We have presented our procedure for verifying LDI of a timed automaton A with non-transient cycles. A main part of our procedure is to translate the automaton into a well-behaved and operational time constrained regular expression ($\Delta Kleene(RG(A'))$). Our achievements can be summarised as follows.

- From the procedure, it is not difficult to see that when the finite upper bound of the LDI is finite (i.e $u < \infty$) the translation procedure ($\Delta Kleene(RG(A'))$) leads to finite TC-REs thus enabling an accurate verification.
- We can also verify the cases $u = \infty$ when the automaton has independent iterations. This property is conserved by Cartesian products. Some identified classes as the Alternating RQ automata fulfill the condition of independent iterations [16], and moreover in general it is also satisfied if the tests in a cycle are always preceded by a reset in the cycle. We can also solve cases of dependent or non transient iterations but this is not covered in this article since we considered them as irrelevant for the exposition of our ideas.
- For $u = \infty$ we can verify conservatively the whole class of TA with non transient cycles.

This is the first proposal that avoids digitisation to verify LDI and that copes with TA as model of real-time systems. Note that since the work is based on the reachability graph construction it could be easily migrated to other formalisms like real-time versions of Petri Nets [4] and ModeCharts [13].

Acknowledgements: We thank Pablo Giambiagi, Gerardo Schneider and Miguel Felder for their useful comments.

References

[1] R. Alur, C. Courcoubetis, and T.A. Henzinger. Computing accumulated delays in real-time systems. In *Proceedings of the 5th International Conference on Computer Aided Verification, CAV'93*, number 697 in Lecture Notes in Computer Science, pages 181–193. Springer-Verlag, 1993.

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

[3] E. Asarin, O. Maler, and P. Caspi. A Kleene theorem for timed automata. In G. Winskel, editor, *Logics In Computer Science*, 1997.

[4] B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using time petri nets. *IEEE Transactions on Software Engineering*, 17(3), March 1991.

[5] V. A. Braberman and Dang Van Hung. On checking timed automata for linear duration invariants. TR 135, UNU/IIST, February 1998,

[6] Zhou Chaochen, C.A.R. Hoare, and A.P. Ravn. A calculus of durations. *Information Processing Letters*, 5(40):269–276, 1991.

[7] Zhou Chaochen, Zhang Jingzhong, Yang Lu, and Li Xiaoshan. Linear duration invariants. In *Formal Techniques in Real-Time and Fault-Tolerant systems*, volume 863 of *Lecture Notes in Computer Science*, 1994.

[8] Li Xuan Dong and Dang Van Hung. Checking linear duration invariants by linear programming. In Joxan Jaffar and Roland H. C. Yap, editors, *Concurrency and Paralellism, Programming, Networking, and Securiry*, number 1179 in Lecture Notes on Computer Science, pages 321–332, December 1996.

[9] M. Hansen. Model-checking discrete duration calculus. *Formal Aspects of Computing*, 6(6A):826–846, Nov-Dec 1994.

[10] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994. Special issue for LICS' 92.

[11] J.E. Hopcroft and J.D. Ulman. *Introduction to Automata Theory, Languages and Computation*. Adison-Wesley, 1979.

[12] Dang Van Hung and Pham Hong Thai. On Checking Parallel Real-time Systems for Linear Duration Invariants. Proceedings of the International Symposium on Software Engineering for Parallel and Distributed Systems (PDSE'98), 20 – 21 April 1998, Kyoto, Japan, Bernd Kramer, Naoshi Uchihira, Peter Croll and Stefano Russo (Eds), IEEE Computer Society Press, 1998, pp. 61 – 71.

[13] F. Jahanian and D. Stuart. A method for verifying properties of modechart specifications. In *Proceedings of the 9th IEEE Real-Time Systems Symposium*, 1988.

[14] I. Kang, I. Lee, and Y.S. Kim. An efficient space generation for the analysis of Real-Time systems. In *Proceedings of the International Symposium on Software Testing and Analisys*, 1996.

[15] Y. Kesten, A. Pnueli, J. Sifakis, and S. Yovine. Integration graphs: a class of decidable hybrid systems. In *Proceedings of Workshop on Theory of Hybrid Systems*, number 736 in Lecture Notes in Computer Science, pages 179–208. Springer-Verlag, June 1992.

[16] W. Lam and R.K. Brayton. Alternating rq timed automata. In C. Courcoubetis, editor, *Proceedings of the 5th Inter. Conference on Computer Aided Verification*, number 697 in Lecture Notes in Computer Science, pages 236–252. Springer-Verlag, June/July 1993.