



The United Nations
University

UNU-IIST

International Institute for
Software Technology

Verification of Linear Duration Invariants by Model Checking CTL Properties

Miaomiao Zhang, Dang Van Hung and Zhiming Liu

June 2008

UNU-IIST and UNU-IIST Reports

UNU-IIST (United Nations University International Institute for Software Technology) is a Research and Training Centre of the United Nations University (UNU). It is based in Macao, and was founded in 1991. It started operations in July 1992. UNU-IIST is jointly funded by the government of Macao and the governments of the People's Republic of China and Portugal through a contribution to the UNU Endowment Fund. As well as providing two-thirds of the endowment fund, the Macao authorities also supply UNU-IIST with its office premises and furniture and subsidise fellow accommodation.

The mission of UNU-IIST is to assist developing countries in the application and development of software technology.

UNU-IIST contributes through its programmatic activities:

1. Advanced development projects, in which software techniques supported by tools are applied,
2. Research projects, in which new techniques for software development are investigated,
3. Curriculum development projects, in which courses of software technology for universities in developing countries are developed,
4. University development projects, which complement the curriculum development projects by aiming to strengthen all aspects of computer science teaching in universities in developing countries,
5. Schools and Courses, which typically teach advanced software development techniques,
6. Events, in which conferences and workshops are organised or supported by UNU-IIST, and
7. Dissemination, in which UNU-IIST regularly distributes to developing countries information on international progress of software technology.

Fellows, who are young scientists and engineers from developing countries, are invited to actively participate in all these projects. By doing the projects they are trained.

At present, the technical focus of UNU-IIST is on **formal methods** for software development. UNU-IIST is an internationally recognised center in the area of formal methods. However, no software technique is universally applicable. We are prepared to choose complementary techniques for our projects, if necessary.

UNU-IIST produces a report series. Reports are either Research **[R]**, Technical **[T]**, Compendia **[C]** or Administrative **[A]**. They are records of UNU-IIST activities and research and development achievements. Many of the reports are also published in conference proceedings and journals.

Please write to UNU-IIST at P.O. Box 3058, Macao or visit UNU-IIST's home page: <http://www.iist.unu.edu>, if you would like to know more about UNU-IIST and its report series.

G. M. Reed, Director



The United Nations
University

UNU-IIST

**International Institute for
Software Technology**

P.O. Box 3058
Macao

Verification of Linear Duration Invariants by Model Checking CTL Properties

Miaomiao Zhang, Dang Van Hung and Zhiming Liu

Abstract

Linear duration invariants (LDI) are important safety properties of real-time systems. They can be easily formulated in terms of a class of chop-free formulas in the Duration Calculus (DC). Compared to other temporal logics, the specification in DC is simpler, neater and more importantly easier to understand. However, directly model checking them is more difficult than model checking properties formulated in the computation tree logic (CTL). In this paper, we present a technique for the verification of the satisfaction of an LDI \mathcal{D} by a timed automaton \mathcal{A} by model checking a CTL property. For this, we construct an untimed automaton G from \mathcal{A} , and prove that \mathcal{A} satisfies \mathcal{D} iff \mathcal{D} is satisfied by the set of all paths of G . To verify that all paths of G satisfy \mathcal{D} , we construct a CTL formula ψ and simply check if G satisfies ψ . By this, we convert the problem of verification of the LDI to the problem of model checking CTL formula. As a result, the CTL model checking techniques and tools, such as UPPAAL, can be used for verification of LDI specified in the DC.

This paper is accepted for presentation and publication in the proceedings ICTAC 2008 in LNCS. The research supported by the project of National Natural Science Foundation of China (No.60603037) and HTTS project funded by Macau Science and Technology Development Fund.

Miaomiao Zhang is a former fellow at UNU-IIST, and now an associate professor at Tongji University (Shanghai). Her research interest is in the area of techniques for verification real-time and fault-tolerant systems. zmm80899@yahoo.com.

Dang Van Hung is a former research fellow, an associate research fellow of UNU-IIST, and now a faculty member of the Vietnam National University. He has been working in the area of real-time embedded systems and their verification. Email: dvh@vnu.edu.vn.

Zhiming Liu is a Senior Research Fellow at UNU-IIST. His research interests include theory of computing systems, emphasising sound methods and tools for specification, verification and refinement of fault-tolerant, realtime and concurrent systems, and formal techniques for OO development. Email: Z.Liu@iist.unu.edu

Contents

1	Introduction	1
2	Preliminary	2
2.1	Timed Automata	2
2.2	Linear Duration Invariants and Duration Properties	3
2.2.1	Duration Calculus	3
2.2.2	Linear Duration Properties and Linear Duration Invariants	4
2.3	Integral Reachability Graph of Timed Automata	5
3	Technique to Check LDI Using CTL	6
3.1	When B is finite	6
3.1.1	The corresponding CTL formula	9
3.2	When B is infinite	10
3.2.1	The procedure for constructing G^+	10
3.3	Verification in UPPAAL	13
4	Case Study	13
5	Conclusion	15

1 Introduction

Linear constraints on the durations of states are important properties of real-time systems. Such a property can be easily formalized as a *chop-free formula* in the Duration Calculus (DC) [11] of the form

$$(1) \quad A \leq \ell \leq B \Rightarrow \sum_{s \in S} c_s \int s \leq M$$

where S is a finite set of system states, $\int s$ is the duration of state s , ℓ denotes the length of the reference time interval, and A , B , and c_s are constants. This class was first introduced and called *linear duration invariants* (LDI) in [12]. The duration $\int s$ of a state s and the length ℓ are mappings from time intervals to reals. For an observation time interval $[b, e]$, $\int s$ defines the accumulated time for the presence of state s over $[b, e]$ and ℓ is the length $e - b$ of the interval. A LDI $A \leq \ell \leq B \Rightarrow \sum_{s \in S} c_s \int s \leq M$ simply says that for any observation time interval $[b, e]$, if the length ℓ of the interval satisfies the constraint $A \leq \ell \leq B$ then the durations of the system states over that interval should satisfy the linear constraint $\sum_{s \in S} c_s \int s \leq M$.

Since timed automata are good models of real-time systems and linear duration invariants are important properties of real-time systems [12], an important problem is whether the verification of a LDI of a timed automaton can be done automatically.

To solve this problem, several algorithms are proposed in the literature, but they have high complexity and they only work with various restrictions either on the timed automata or on the LDIs [6, 12, 7, 3]. For improving the complexity the algorithms proposed in [10, 8, 9] are restricted to the class of the so-called *discretisable properties*. A property is discretisable iff it is satisfied by all the behaviors of a timed automaton exactly when it is satisfied by the *integral behaviors* of the automaton (i.e. behavior in which transitions take place only at integer time). Furthermore, to the best of our knowledge, there are no existing implementations of these algorithms.

Another popular logic for specification and verification of real-time systems is the *computation tree logic* (CTL)[5]. Effective techniques and tools [14, 15] have been developed for checking real-time systems modeled by timed automata [1, 2] against properties specified in CTL. This motivates our interest in this paper to study the possibility of reducing the problem of verification of a LDI of a timed automaton to an equivalent problem of model checking a CTL property of an automaton. The goal is that, instead of directly checking a LDI \mathcal{D} for a timed automata using the techniques in [10, 8, 9], we check a CTL property for a translated model G , using popular model checker, such as UPPAAL.

Our technique following the idea proposed in [9] for checking \mathcal{D} for an automaton \mathcal{A} , we construct a *reachability graph* \mathcal{RG} . We then optimize \mathcal{RG} to derive a graph G . However, different in technique from [9] that uses extra sub-vertices to record the duration that the system stays in

a node of \mathcal{RG} , we use an integer variable n to denote the duration that the system stays in a vertex v of \mathcal{RG} together with a self-loop transition of v . This makes our model much simpler and thus easier for model checking. Also, we use a variable gc to bind the value of observation time, and another variable d to bind the value of duration of system states. The value of n is bounded because we remove infinite edges in \mathcal{RG} . In achieving these, our graph is constructed carefully in different ways depending on whether the constant B in formula (1) is finite or not.

We use $\mathcal{P}(\mathbf{G})$ to denote the set of all paths of the graph \mathbf{G} constructed from \mathcal{A} . We then prove that \mathcal{D} is satisfied by \mathcal{A} iff \mathcal{D} is satisfied by the paths $\mathcal{P}(\mathbf{G})$, i.e. $\mathcal{A} \models \mathcal{D}$ if and only if $\mathcal{P}(\mathbf{G}) \models \mathcal{D}$. Finally, we define a CTL formula ψ for \mathbf{G} , and prove that $\mathcal{P}(\mathbf{G}) \models \mathcal{D}$ iff $\mathbf{G} \models \psi$.

The rest of the paper is organized as follows. Section 2 recalls some basic notions of timed automaton and Duration Calculus. It also introduces the integral reachability graph of timed automaton. The main technical contribution is presented in Section 3. There, we introduce two kinds of graphs respectively for the cases when B is finite and when B is infinite, and prove the main theorems. We then show an algorithm for checking an CTL of a graph of the automaton. A case study is given in Section 4 to illustrate our technique. Conclusions are discussed in Section 5.

2 Preliminary

We introduce the notions that we need in this paper. These include *timed automata*, *region graphs*, and Linear Duration Invariants (LDI) defined in DC.

2.1 Timed Automata

We first recall the definition of timed automata given in in [1, 2] and explain their behavior. For this, we use $\mathbf{R}^{\geq 0}$ and \mathbf{N} to denote the sets of nonnegative real numbers and natural numbers, respectively.

A timed automaton is a finite state machine equipped with a set of clocks. We use a set X of real value variables to resents the clocks and let $\Phi(X)$ be the set of clock constraints on X , which are conjunctions of the formulas of the form $x \leq c$ or $c \leq x$, where $x \in X$ and $c \in \mathbf{N}$.

Definition 1 *A timed automaton \mathcal{A} is a tuple $\langle L, s_0, \Sigma, X, E, I \rangle$, where*

- L is a finite set of locations,
- $s_0 \in L$ is the initial location,
- Σ is a finite set of symbols (action names),

- X is a finite set of clocks,
- I is a mapping that assigns each location $s \in L$ with a clock constraint $I(s) \in \Phi(X)$ called the invariant of location s . Intuitively, the timed automaton only stays at s when the values of the clocks satisfy the invariant $I(s)$.
- $E \subseteq L \times \Phi(X) \times \Sigma \times 2^X \times L$ is a set of switches. A switch $\langle s, \varphi, a, \lambda, s' \rangle$ represents a transition from location s to location s' with event a , where φ is a clock constraint over X that specifies the enabling condition of the switch, and $\lambda \subseteq X$ is the set of clocks to be reset to 0 by this switch.

A clock interpretation ν is a mapping that assigns a nonnegative real value to each clock in X . For $\delta \in \mathbf{R}^{\geq 0}$, let $\nu + \delta$ denote the clock interpretation which maps every clock $x \in X$ to the value $\nu(x) + \delta$. For $\lambda \subseteq X$, let $\nu[\lambda := 0]$ denote the clock interpretation which assigns 0 to each $x \in \lambda$ and agrees with ν over the rest of the clocks.

A state of automaton \mathcal{A} is a pair (s, ν) where s is a location of \mathcal{A} and ν is a clock interpretation which satisfies the invariant $I(s)$. State (s_0, ν_0) is the initial state where s_0 is the initial location of \mathcal{A} and ν_0 is the clock interpretation for which $\nu_0(x) = 0$ for all clocks x . Here we should note that $I(s_0)$ is always assumed to be satisfied by ν_0 to make the automaton \mathcal{A} operate.

2.2 Linear Duration Invariants and Duration Properties

2.2.1 Duration Calculus

DC [11] is a logic for reasoning about durations of states of real-time systems of real-time systems. A comprehensive introduction to *DC* is given in the monograph by Zhou and Hansen [13]. In DC, a state s is *interpreted* as a function from the time domain $\mathbf{R}^{\geq 0}$ to the boolean values $\{1, 0\}$, and s is 1 at time t if the system is in state s and 0 otherwise. Therefore, a model of DC formula consists of an interpretation \mathcal{I} of the states and a time interval $[b, e]$. It represents an observation of the behavior of the system in term of presence and absence of the states in the interval of time. Given an interpretation \mathcal{I} , the duration of a state s over the time interval $[b, e]$ is defined as the integral $\int_b^e \mathcal{I}_s(t) dt$, which is exactly the accumulated present time of s in the interval $[b, e]$ under the interpretation \mathcal{I} .

We consider the set of DC models that express all the observations of the behaviors of a timed automaton. Each behavior $\rho = (s_0, t_0)(s_1, t_1) \dots$ of timed automaton \mathcal{A} defines an interpretation \mathcal{I} of the DC formulas about the states of \mathcal{A} : for any state s of \mathcal{A} , $\mathcal{I}_s(t) = 1$ iff $\exists i \bullet (s_i = s \wedge t \in [t_i, t_{i+1}))$. We also denote such \mathcal{I} by (\bar{s}, \bar{t}) where $\bar{s} = (s_0, s_1, \dots)$ and $\bar{t} = (t_0, t_1, \dots)$ are respectively the sequence of states s_i and the sequence of time stamps t_i from the behavior ρ . Hence, $(\bar{s}, \bar{t}, [b, e])$ is also considered as a DC model representing the observation of \mathcal{A} in the time interval $[b, e]$, which is a possible observation of the timed automaton \mathcal{A} over interval $[b, e]$. For this reason, we also call $(\bar{s}, \bar{t}, [b, e])$ an \mathcal{A} -model of DC.

Let $\mathcal{M}(\mathcal{A})$ denote the set of \mathcal{A} Models of DC. For a given timed automaton \mathcal{A} , the following three classes of \mathcal{A} models of DC are identified and studied in [9]:

1. the set of all \mathcal{A} models that start from time 0 and end at any time point

$$\mathcal{M}_0(\mathcal{A}) \hat{=} \{\sigma \mid \sigma = (\bar{s}, \bar{t}, [0, T]) \in \mathcal{M}(\mathcal{A}), T \geq 0\}$$

2. the models representing the observations starting and ending at those time points at which the automaton \mathcal{A} switches from one location to another location:

$$\mathcal{M}_d(\mathcal{A}) \hat{=} \{\sigma \mid \sigma = (\bar{s}, \bar{t}, [t_u, t_v]) \in \mathcal{M}(\mathcal{A}), t_u, t_v \text{ occur in } \bar{t} \text{ and } t_u \leq t_v\}$$

3. the models with integral observation intervals

$$\mathcal{M}_I(\mathcal{A}) \hat{=} \{\sigma \mid \sigma = (\bar{s}, \bar{t}, [b, e]) \in \mathcal{M}(\mathcal{A}) \text{ and } b, e \in \mathbf{N}, b \leq e\}$$

2.2.2 Linear Duration Properties and Linear Duration Invariants

A *linear duration invariant (LDI)* of a timed automaton $\mathcal{A} = \langle L, s_0, \Sigma, X, E, I \rangle$ is a DC formula \mathcal{D} of the form

$$A \leq \ell \leq B \Rightarrow \sum_{s \in L} c_s \int s \leq M$$

where c_s , A , B and M are real numbers, $A \leq B$ (B may be ∞), the DC term $\int s$ denotes the duration of location s , and the DC term ℓ the length of the interval. A LDI \mathcal{D} is evaluated in an \mathcal{A} model $(\mathcal{I}, [b, e])$ to tt , denoted by $(\mathcal{I}, [b, e]) \models \mathcal{D}$, iff $A \leq e - b \leq B \Rightarrow \sum_{s \in L} c_s \int_b^e \mathcal{I}_s(t) dt \leq M$ holds. \mathcal{D} is satisfied by \mathcal{A} , denoted by $\mathcal{A} \models \mathcal{D}$, if $(\mathcal{I}, [b, e]) \models \mathcal{D}$ holds for any model $(\mathcal{I}, [b, e]) \in \mathcal{M}(\mathcal{A})$. We use $\Sigma(\mathcal{D})$ to denote the the sum of the durations $\sum_{s \in L} c_s \int s$.

A LDI \mathcal{D} of a timed automaton \mathcal{A} is said to be *discretisable* with respect to \mathcal{A} iff $\mathcal{A} \models \mathcal{D}$ exactly when $\mathcal{M}_I(\mathcal{A}) \models \mathcal{D}$. We have the following theorem [9].

Theorem 1 *Let $\mathcal{D} \hat{=} A \leq \ell \leq B \Rightarrow \sum_{s \in L} c_s \int s \leq M$ be a LDI of a timed automaton \mathcal{A} . It is discretisable with respect to \mathcal{A} if A and B are integers. Here we consider ∞ as an integer.*

2.3 Integral Reachability Graph of Timed Automata

An integral reachability graph $\mathcal{RG} = (\mathbf{V}_R, \mathbf{E}_R)$ of a timed automaton \mathcal{A} is constructed as follows. Each vertex $\mathbf{v} \in \mathbf{V}_R$ will be a pair $\langle s, \pi \rangle$, where s is a state of \mathcal{A} , and π is an integral region of \mathcal{A} , i.e. the restriction on the set of integers of a clock region of \mathcal{A} (see [9] for more detailed definition). \mathbf{E}_R is initialized to \emptyset , and \mathbf{V}_R is initialized to $\{\langle s_0, \pi_0 \rangle\}$, where s_0 is initial location of \mathcal{A} and π_0 is the integral region containing the assignment that assigns 0 to all clocks, i.e. π_0 contains only the assignment that assigns 0 to all clocks. Then, \mathbf{V}_R is expanded as follows. If a vertex $\langle s, \pi \rangle \in \mathbf{V}_R$ has a successor $\langle s', \pi' \rangle$, i.e. there exist $d \geq 0$ and an transition $e = \langle s, \varphi, a, \lambda, s' \rangle$ such that $(s, \nu) \xrightarrow{d, a} (s', \nu')$, then $\langle s', \pi' \rangle$ is added into \mathbf{V}_R and $e = (\langle s, \pi \rangle, \langle s', \pi' \rangle)$ is an edge in \mathbf{E}_R . $[\mathbf{l}(e), \mathbf{u}(e)]$, where $\mathbf{l}(e)$ and $\mathbf{u}(e)$ are the minimal and maximal integer time delay that the automaton can stay at location s before it transits into location s' . $\mathbf{l}(e)$ and $\mathbf{u}(e)$ are defined as:

$$\begin{aligned} \mathbf{l}(e) &= \inf \left\{ d \geq 0 \mid d \in \mathbf{N}, \langle s, \pi \rangle \xrightarrow{d, a} \langle s', \pi' \rangle \right\}, \\ \mathbf{u}(e) &= \sup \left\{ d \geq 0 \mid d \in \mathbf{N}, \langle s, \pi \rangle \xrightarrow{d, a} \langle s', \pi' \rangle \right\}. \end{aligned}$$

A detailed description of the algorithm are given in [10, 9]. From the definition of $\langle s, \pi \rangle$ and $\langle s', \pi' \rangle$, we have $\mathbf{l}(e) \leq \mathbf{u}(e)$ and it is possible that $\mathbf{u}(e) = \infty$. We call e an infinite edge if $\mathbf{u}(e) = \infty$. We will label an edge e by $(\mathbf{v}, \mathbf{v}', [\mathbf{l}(e), \mathbf{u}(e)])$.

There is a clear correspondence relation between the paths of the graph and the \mathcal{A} models of DC [9]. Let $\mathbf{p} = \mathbf{v}_1 \mathbf{v}_2 \dots \mathbf{v}_m$ be a path in \mathcal{RG} and $\bar{d} = d_1, d_2, \dots, d_{m-1}$ a sequence of integers, where $d_i \in [\mathbf{l}(\mathbf{v}_i, \mathbf{v}_{i+1}), \mathbf{u}(\mathbf{v}_i, \mathbf{v}_{i+1})]$, for $i = 1..m-1$. We call the sequence $\wp = \mathbf{v}_1 d_1 \mathbf{v}_2 d_2 \dots \mathbf{v}_{m-1} d_{m-1} \mathbf{v}_m$ (written as $\wp = (\mathbf{p}, \bar{d})$ for short) *weighted interpretation* of \mathbf{p} .

We define the following

- the *length* of \wp : $l(\wp) \hat{=} \sum_{i=1}^{m-1} d_i$
- the *cost* of \wp : $\theta(\wp) \hat{=} \sum_{i=0}^{m-1} c_{\mathbf{v}_i} d_i$ where $c_{\mathbf{v}_i}$ is the coefficient c_{s_i} in LDI \mathcal{D} when s_i is the location of \mathbf{v}_i .
- a weighted interpretation \wp is said to satisfy \mathcal{D} , denoted by $\wp \models \mathcal{D}$, iff

$$A \leq l(\wp) \leq B \Rightarrow \theta(\wp) \leq M$$

- a graph \mathcal{RG} is said to satisfy LDI \mathcal{D} , denoted by $\mathcal{RG} \models \mathcal{D}$, iff $\wp \models \mathcal{D}$ for all weighted interpretations \wp of \mathcal{RG} .

The fact that $\mathcal{M}_d(\mathcal{A}) \models \mathcal{D}$ iff $\mathcal{RG} \models \mathcal{D}$ can be derived from relation in the following lemma between the models in $\mathcal{M}_d(\mathcal{A})$ and the weighted interpretations of \mathcal{RG} .

Lemma 1 *For any model $\sigma \in \mathcal{M}_d(\mathcal{A})$, there exists a weighted interpretation \wp of \mathcal{RG} such that $l(\sigma) = l(\wp)$ and $\theta(\sigma) = \theta(\wp)$, and vice versa.*

Case analysis is needed when checking a region graph with infinite edges. This is done following the two lemmas below, in which $A \leq \ell \leq B$ is the premise of the LDI \mathcal{D} of concern. For a node $v = \langle s, \pi \rangle$ we simply write c_v for the coefficient c_s in the LDI \mathcal{D} .

Lemma 2 *Assume that $e = (v, v', [l(e), \infty))$ is an infinite edge of a region graph \mathcal{RG} . $\mathcal{RG} \not\models \mathcal{D}$ if $B = \infty$ and $c_v > 0$.*

Lemma 3 *Assume that $e = (v, v', [l(e), \infty))$ is an infinite edge of \mathcal{RG} . Then the label $[l(e), \infty)$ can be replaced as follows without affecting the result of checking $\mathcal{RG} \models \mathcal{D}$.*

- If $B = \infty$ and $c_v \leq 0$, replace $[l(e), \infty)$ by $[l(e), u(e)]$ with $u(e) = \max\{l(e), A\}$.
- If $B < \infty$, replace $[l(e), \infty)$ by $[l(e), u(e)]$ with $u(e) = \max\{l(e), B\}$.

Therefore, to verify $\mathcal{M}_d(\mathcal{A}) \models \mathcal{D}$, lemma 2 allows us to conclude with $\mathcal{M}_d(\mathcal{A}) \not\models \mathcal{D}$ immediately if the conditions of the lemma hold, otherwise we can use lemma 3 to translate the graph to one without infinite edges. In the rest of the paper, we assume that \mathcal{RG} does not contain infinite edges.

3 Technique to Check LDI Using CTL

We now present our technique to reduce the verification of the satisfaction of a LDI by a timed automaton to checking a CTL formula of timed automaton. From the discussion in the previous section, we only need to construct a graph G with variables from a reachability graph \mathcal{RG} (without infinite edges) and a CTL formula ψ such that $\mathcal{RG} \models \mathcal{D}$ if and only if $G \models \psi$. We distinguish two cases of the constant B in the premise of \mathcal{D} : 1) B is finite, 2) B is infinite.

3.1 When B is finite

We first introduce integer variables n , gc and d

- n is used to count the number of time units that \mathcal{RG} stays in a vertex v before moving to another vertex,

- gc is used to record the time length of an observation interval (corresponding to a path in \mathcal{RG}), and
- d records the sum of durations of states.

All of the variables are initialized to 0. For the reachability graph $\mathcal{RG} = (\mathbf{V}_R, \mathbf{E}_R)$, an untimed graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ with integer variables n , gc and d is constructed by a procedure.

For the description of the procedure, we need the following normalization function for variable gc .

Definition 2 ($B + 1$ -normalization)

$$\text{norm}_{B+1}(gc) = \begin{cases} gc + 1, & \text{if } gc \leq B \\ B + 1, & \text{if } gc > B \end{cases}$$

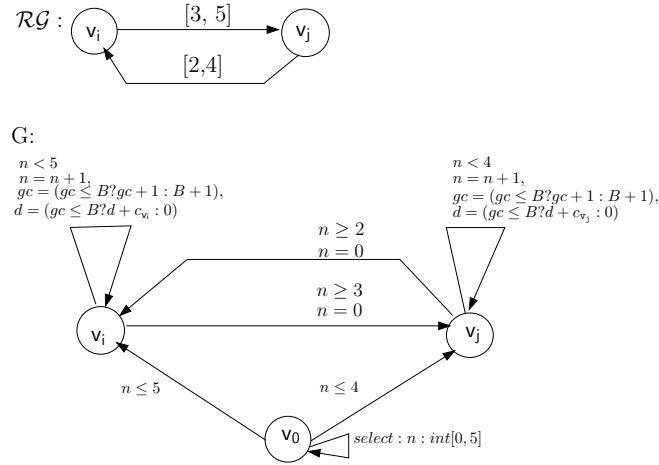
The intuitive intention is that gc records the length of the current observation interval, and the LDI \mathcal{D} is satisfied trivially when it exceeds the constant B in premise of \mathcal{D} . Hence, we do not need to record every value of gc that bigger than B . It is sufficient to record $B + 1$ when the length of the observation time exceeds B . The procedure for construction $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ from $\mathcal{RG} = (\mathbf{V}_R, \mathbf{E}_R)$ is given as follows. In the construction, we introduce an extra vertice v_0 as the single initial node to set the initial value of data variavles. Note that with data variables, a CTL formula is not a state formula in our system graph because with different histories leading to a node, a data variable might have different values.

Step 1. $\mathbf{V} := \mathbf{V}_R \cup \{v_0\}$, $\mathbf{E} := \mathbf{E}_R \cup \{(v_0, v) | v \in \mathbf{V}_R\}$, where v_0 is a fresh node and is also considered as an initial node in \mathbf{G} .

1. Let $u(v)$ be the maximum time units that \mathcal{RG} stays in v . In the edge (v_0, v) , we have the guard $n \leq u(v)$.
2. Let $T = \max \{u(v) | v \in \mathbf{V}_R\}$. In v_0 there is a self-loop transition that nondeterministically select the value of n between $[0, T]$. In UPPAAL, this can be described by a **select** language $n : \text{int}[0, T]$.

Starting with the initial node v_0 and the initial values $gc, n, d := 0, 0, 0$, the above two conditions imply that gc starts to count time from the first enter of any node v ; and the system can stay the node v for any n time units provided that $n \leq u(v)$.

Step 2. For each edge $e = ((v_i, v_j), [l(e), u(e)]) \in \mathbf{E}_R$

Figure 1: “Discretising” graph when B is finite

1. $E := E \setminus \{e\}$,

2. $E := E \cup E1 \cup E2$, where

- $E1 := \{(v_i, v_i)\}$, and for this edge we have the guard $\varphi : n < u(e)$ and the multiple assignment λ :
 - $n := n + 1$,
 - $gc := (gc \leq B?gc + 1 : B + 1)$,
 - $d := (gc \leq B?d + c_{v_i} : 0)$

The second assignment assigns gc the value of $gc + 1$ if $gc \leq B$ and the value $B + 1$ otherwise, i.e. it is the implementation of the $B + 1$ normalization. Similarly, the third assignment assigns d the value of $d + c_{v_i}$ if $gc \leq B$ and 0 otherwise.

- $E2 := \{(v_i, v_j)\}$, and for this edge we have the guard $\varphi : n \geq l(e)$ and the assignment $\lambda : n := 0$,

Notice that precisely speaking an edge in G is labeled with a guard and a set of assignments. There can be different edges between the same pair of nodes but with different labels. We call G constructed by this procedure the *untimed graph* of \mathcal{A} for \mathcal{D} .

Roughly speaking, G is built by adding self-loop edges in vertex v_i and the edges between v_i and v_j . By assigning 0 to d when $gc > B$, the value of variable d is finite. Besides, gc is bounded by $B + 1$. Since \mathcal{RG} does not contain infinite edges, the value of n is bounded. This construction is much simpler than the one in paper [9], which “splitting” each edge $e = (v, v', [l(e), u(e)])$ of \mathcal{RG} into $u(e)$ small edges with the length (weight) 1 by adding $u(e) - 1$ sub-vertices. Figure 1 gives an example how to build the graph G from the graph \mathcal{RG} .

3.1.1 The corresponding CTL formula

We now define the CTL ψ_1 formula corresponding to the LDI \mathcal{D} of a timed automaton \mathcal{A}

$$(2) \quad \psi_1 \hat{=} \mathbf{A}[\Box \text{ not } A \leq gc \leq B \wedge d > M]$$

We call ψ_1 the *CTL-version* of \mathcal{D} for \mathcal{A} .

Lemma 4 *Let \mathcal{D} be a LDI of a timed automaton \mathcal{A} , \mathbf{G} the untimed graph of \mathcal{A} for \mathcal{D} , and ψ_1 the CTL-version of \mathcal{D} for \mathcal{A} . Then there exists a path $\mathbf{p} \in \mathcal{P}(\mathbf{G})$ such that $\mathbf{p} \not\models \psi_1$ iff there exists an integral model $\sigma \in \mathcal{M}_I(\mathcal{A})$ such that $\sigma \not\models \mathcal{D}$.*

Proof: From the construction procedure for \mathbf{G} , there is an obvious one-to-one correspondence between a path ρ of \mathcal{RG} and a path ρ_g of \mathbf{G} starting from the node v_0 , that represents an observation of the system in the two models. Let $\ell(\rho)$ be the length of ρ , which represents the time of the observation, and $last(\rho_g)$ be the last node of ρ_g .

1. When $\ell(\rho) \leq B$, the value of gc at $last(\rho_g)$ equals $\ell(\rho)$, and the value of d at $last(\rho_g)$ is the value of the sum $\Sigma(\mathcal{D})$.
2. When $\ell(\rho) > B$, the value of gc at $last(\rho_g)$ is $B + 1$.

Consequently, the lemma follows immediately from the definition of the satisfaction relations \models for LDI and CTL formulas.



From this lemma and the discretisability of LDI, we have to check only integer models of the automaton \mathcal{A} . Hence we can restrict ourselves to the integral region graph. The theorem below follows straightforward.

Theorem 2 *When B is finite, the verification of a LDI \mathcal{D} by a timed automaton \mathcal{A} is equivalent to the verification of the satisfaction of CTL-version of \mathcal{D} for \mathcal{A} by the set of paths $\mathcal{P}(\mathbf{G})$, i.e. $\mathcal{A} \models \mathcal{D}$ if and only if $\mathcal{P}(\mathbf{G}) \models \psi_1$.*

We can use a model checker for CTL, such as UPPAAL, to verify $\mathcal{P}(\mathbf{G}) \models \psi_1$.

3.2 When B is infinite

In terms of the graph constructed as above, for the case that B is infinite, gc can increase infinitely and d can take an arbitrary value. This case makes it impossible to check the property \mathcal{D} using the technique presented in the previous section. To bind the value of gc we will use “ A -normalization” as follows in the graph construction, where A is the other constant in the premise of the LDI.

Definition 3 (*A-normalization*)

$$\text{norm}_A(gc) = \begin{cases} gc + 1, & \text{if } gc < A \\ A, & \text{if } gc \geq A \end{cases}$$

Intuitively, the A normalization is dual to the B -normalization. The variable gc is still used for the length of the observation. Therefore with this normalization, for checking LDI \mathcal{D} when gc equals A , we only need to check whether there exists a path along which the value of $\Sigma(\mathcal{D})$ is bigger than M . Now we introduce a number to bound the value of d .

Definition 4 Let V^+ be the set of all nodes v_p in \mathcal{RG} for which $c_{v_p} > 0$. Then we call the value $Q = \sum_{v_p \in V^+} (c_{v_p} \cdot u(v_p))$ the maximum increment of \mathcal{RG} .

The intended meaning for the number Q is that in case there is no loop in a path of \mathcal{RG} , the value of d along that path can increase at most Q . In other words, if the value of d along a path increases more than Q , then there must be a positive loop in the path. The graph $G^+ = (V, E)$ is constructed from $\mathcal{RG} = (V_R, E_R)$ in a way similar to the case that B is finite by the following procedure. For any $v \in V$, we make variable d bounded by updating it differently depending on whether the coefficient c_v in $\Sigma(\mathcal{D})$ is negative or not.

3.2.1 The procedure for constructing G^+

Step 1. This step is the same as that in the construction of G .

Step 2. For each edge $e = ((v_i, v_j), [l(e), u(e)]) \in E_R$, where v_i has a non-negative coefficient c_{v_i} , do the following:

1. $E := E \setminus \{e\}$,

2. $E := E \cup E1 \cup E2$, where

- $E1 := \{(v_i, v_i)\}$, and in this edge we have the guard $\varphi : n < u(\mathbf{e})$ and the multiple assignment λ :
 - $n := n + 1$,
 - $gc := (gc < A ? gc + 1 : A)$,
 - $d := (gc \geq A \wedge d > M ? M + 1 : d + c_{v_i})$
- $E2 := \{(v_i, v_j)\}$, and in this edge we have the guard $\varphi : n \geq l(\mathbf{e})$ and the assignment $\lambda : n := 0$.

Step 3. For each edge $\mathbf{e} = ((v_i, v_j), [l(\mathbf{e}), u(\mathbf{e})]) \in E_R$, where v_i has a negative coefficient c_{v_i} , do the following:

1. $E := E \setminus \{\mathbf{e}\}$,

2. $E := E \cup E1 \cup E2$, where

- $E1 := \{(v_i, v_i)\}$, and in this edge we have the guard $\varphi : n < u(\mathbf{e})$ and the assignment λ :
 - $n := n + 1$,
 - $gc := (gc < A ? gc + 1 : A)$,
 - $d := (gc \geq A \wedge d < M - Q ? d : d + c_{v_i})$
- $E2 := \{(v_i, v_j)\}$, and in this edge we have the guard $\varphi : n \geq l(\mathbf{e})$ and the assignment $\lambda : n := 0$,

In case of c_{v_i} is non-negative, when $gc \geq A$ and $d > M$, by setting d to $M + 1$, the value of d is finite. Moreover, when $gc \geq A$, gc remains as A , so gc is a bounded variable. Since the states that satisfy $gc \geq A \wedge d = M + 1$ imply $\mathcal{P}(\mathbf{G}^+) \not\models \mathcal{D}$, it is obvious that the update does not change the verification result.

When c_{v_i} is negative, the edge of the graph from v_i to v_i is the same as that of the non-negative one, except that the value update of d is $d := (gc \geq A \wedge d < M - Q ? d : d + c_{v_i})$. It is not hard to see why we set d to $d + c_{v_i}$ if $\neg(gc \geq A \wedge d < M - Q)$: we have to evaluate the value of d precisely when we do not have enough information for verifying if \mathcal{D} is satisfied. Now we prove that if $gc \geq A \wedge d < M - Q$, the value of d remaining unchanged does not alter the checking result of the LDI. To do so, we define another graph \mathbf{G}^\bullet that is the same as \mathbf{G}^+ except that if $gc \geq A \wedge d < M - Q$ the assignment for d is $d := d + c_{v_i}$.

Similar to the case that B is finite, we define a CTL-version of \mathcal{D} , denoted by ψ_2 , for a timed automaton \mathcal{A} .

(3) $\psi_2 : \mathbf{A}[] \text{ not } gc \geq A \wedge d > M$.

Lemma 5 *There exists a path $\rho \in \mathcal{P}(G^+)$ such that $\rho \not\models \psi_2$ if and only if there exists a path $\rho' \in \mathcal{P}(G^\bullet)$ such that $\rho' \not\models \psi_2$.*

Proof: Notice that the topological structure of G^+ and G^\bullet are the same. Each path $\rho = v_0^+, \dots, v_m^+$ in G^+ corresponds to exactly one path $\rho^\bullet = v_0^\bullet, \dots, v_m^\bullet$ in G^\bullet . Let v_i^+ and v_i^\bullet be any two corresponding nodes respectively in ρ and ρ^\bullet . Then the value of gc at vertex v_i^+ is the same as the value of that at vertex v_i^\bullet . Due to the different updates of d in ρ and ρ^\bullet for the negative coefficient of a vertex, we know that at vertex v_i^+ , the value of d is bigger than or equal to the value of d at v_i^\bullet . Hence, a path $\rho' = \rho^\bullet$ in G^\bullet that does not satisfy ψ_2 then its corresponding path ρ in G^+ does not satisfy ψ_2 .

To prove the other direction, let ρ in G^+ be such that $\rho \not\models \psi_2$ and ρ starts from the initial node v_0 . If $\rho^\bullet \not\models \psi_2$, we are done. Otherwise, we need to show that there will be a “positive cycle” in ρ , i.e. there is a cycle such that going along the cycle will increase the value of d properly by at least 1. We now give the illustration for the case $\rho \not\models \psi_2 \wedge \rho^\bullet \models \psi_2$. This case denotes that the values of d on ρ and on ρ^\bullet are different and there should be a first node v_j^+ along ρ where the condition $gc \geq A \wedge d < M - Q \wedge c_{v_j} < 0$ holds. Thus, from v_j^+ , the value of d is increased by at least $Q + 1$ to make $\rho \not\models \psi_2$.

From the definition of Q , in ρ there must be a “positive cycle” along which d will be increased by at least 1. From the correspondence relation between ρ and ρ^\bullet , ρ^\bullet must also have a positive cycle \mathcal{C} . Thus ρ' is formed by increasing the number of repetition of the cycle \mathcal{C} in ρ^\bullet , such that $\rho' \not\models \psi_2$.

♣

Therefor we conclude that d is a bounded integer variable. Figure 2 gives an example how to build the graph G^+ from the graph \mathcal{RG} when c_{v_i} is non-negative and c_{v_j} is negative. The lemma bellow follows from the definitions of relations \models for \mathcal{D} and ψ_2 .

Lemma 6 *Given a timed automaton \mathcal{A} and a LDI \mathcal{D} . Then there exists a path $\rho \in \mathcal{P}(G^+)$ such that $\rho \not\models \psi_2$ iff there exists an integral model $\sigma \in \mathcal{M}_I(\mathcal{A})$ such that $\sigma \not\models \mathcal{D}$.*

Now we have our main theorem for the case when B is infinite.

Theorem 3 *When B is infinite, the verification of a LDI \mathcal{D} by a timed automaton \mathcal{A} is equivalent to the verification of the satisfaction of CTL-version ψ_2 of \mathcal{D} for \mathcal{A} by the set of paths $\mathcal{P}(G^+)$, i.e. $\mathcal{A} \models \mathcal{D}$ if and only if $\mathcal{P}(G^+) \models \psi_2$.*

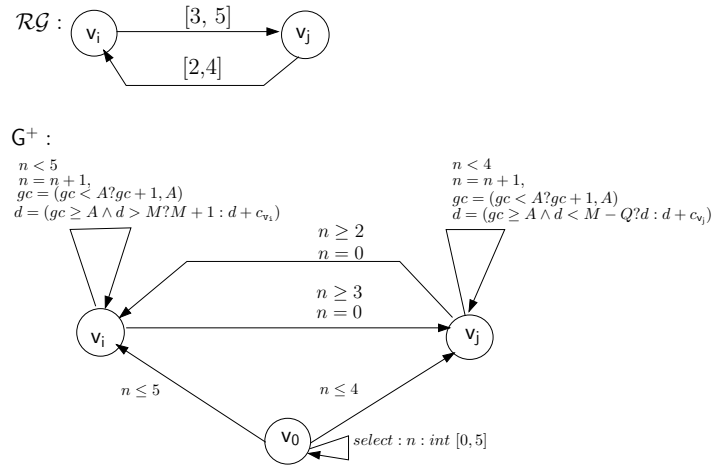


Figure 2: “Discretising” graph when B is infinite and c_{v_i} is non-negative, c_{v_j} is negative

3.3 Verification in UPPAAL

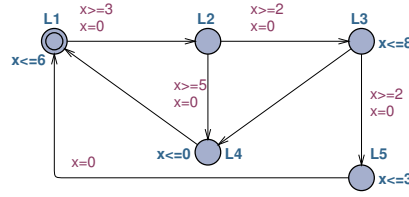
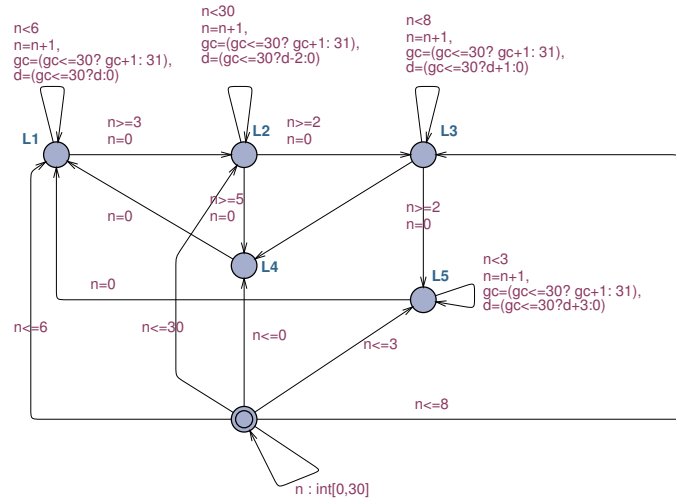
The model checking tool-UPPAAL that is available at www.uppaal.com, is an integrated tool environment for formal specification, validation and verification of real time systems modeled as networks of timed automata. UPPAAL uses a simplified version of CTL to express the requirement specification. The query language consists of path formulas and state formulae. State formulae describe individual states, whereas path formulae quantify over paths or traces of the model. Path formulae can be classified into reachability, safety and liveness.

The properties ψ_1 and ψ_2 in our checking algorithm are safety properties. This implies that on one hand, we can draw the graph in UPPAAL and just click the “check” button to verify the safety property ψ_1 or ψ_2 from the extra vertex v_0 . And this is done automatically. On the other hand, since we use the same modeling language and the same query language used by UPPAAL, the checking algorithm can be easily implemented in UPPAAL.

4 Case Study

In [11], the Duration Calculus is used to prove that a gas burner does not leak excessively. That is, the accumulated time of leakage is at most one twentieth of the time in any interval of at least 60 seconds. Following the techniques in Section 3, using UPPAAL, we have checked that the LDI property is satisfied. We now use a more general model \mathcal{A} shown in Fig 3 to illustrate our techniques.

The LDI properties to be checked are:

Figure 3: A Model \mathcal{A} Figure 4: “Disretising” graph \mathcal{A}'

1. $D_1 : 10 \leq \ell \leq 30 \rightarrow (-2 \times \int L2 + \int L3 + 3 \times \int L5) \leq 30.$
2. $D_2 : 10 \leq \ell \leq \infty \rightarrow (-2 \times \int L2 + \int L3 + 3 \times \int L5) \leq 30.$

Let e_1, e_2 be the infinite edges from $L2$ respectively to $L3$ and $L4$. To check whether or not D_1 is satisfied, we first use the methods of removing infinite edge in subsection 2.3, to translate e_1 and e_2 to finite edges. We thus have $l(e_1) = 2, l(e_2) = 5, u(e_1) = u(e_2) = 30$. Also, in this case we have $A = 10, B = 30, M = 30$. The CTL formula C_1 for D_1 is:

$$(4) \quad C_1 : \mathbf{A} \square \text{ not } (gc \geq 10 \ \&\& \ gc \leq 30 \ \&\& \ d > 30)$$

In terms of the technique in subsection 3.2, we construct the graph \mathcal{A}' shown in Fig.4 that is used in UPPAAL as a model to check C_1 . The checking result shows that $\mathcal{A}' \models C_1$. Therefore, we have $\mathcal{A} \models D_1$.

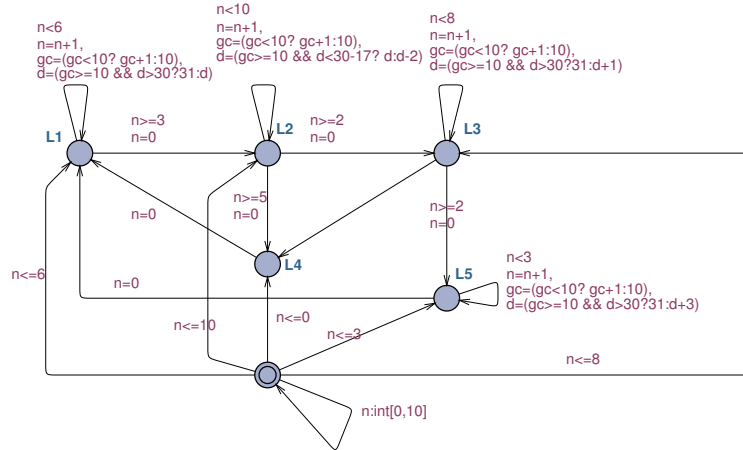


Figure 5: “Discretising” graph \mathcal{A}''

We now check D_2 and construct the model \mathcal{A}'' shown in Fig 5. In this case, $u(e_1) = u(e_2) = 10$, $A = 10$, $B = \infty$, $M = 30$, $Q = 8 + 3 \times 3 = 17$. The CTL logic C_2 for D_2 is:

$$(5) \quad C_2 : A[] \text{ not } (gc \geq 10 \ \&\& \ d > 30)$$

We have checked with UPPAAL, and the checking result is that $\mathcal{A}'' \not\models C_2$. Thus $\mathcal{A} \not\models D_2$.

5 Conclusion

The examples in the paper show that the DC formulation of Linear Duration Constraints is simpler, neater and easy to understand than those in LTL and CTL. The automata that are given for the DC specification are simpler than those that would be constructed for a LTL or CTL specification. However, the existing algorithms for model checking Linear Duration Calculus invariants are complex and have not been implemented. A lot of works have been done recently [16, 17, 18] for the development of model checking tools for Duration Calculus formulas. In spite of some model checking tools for Duration Calculus formulas are available now, to our knowledge, compared with other temporal logics, the techniques developed for DC are still not widely applicable in industrial fields. In this paper, we have presented a different approach to the problem. Our approach is to reduce the verification of a LDI to model checking CTL. This allows us to use or easily extend the techniques in the current tools for CTL, such as UPPAAL and SMV, to check linear Duration Calculus Invariant of real-time embedded systems. Furthermore, since the CTL formulas ψ_1 and ψ_2 can directly be specified in the linear time logic LTL, we thus can use the model checkers like SPIN to check the LDI. We believe that this technique will help to make Duration Calculus more applicable in industry field.

References

- [1] R. Alur and D.L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, pages 183–235, 1994.
- [2] R. Alur. Timed Automata. *Proceedings of 11th International Conference on Computer-Aided Verification*, LNCS 1633, pages: 8-22, Springer Verlag, 1999.
- [3] Victor A. Braberman and Dang Van Hung. On Checking Timed Automata for Linear Duration Invariants. In: *Proceedings of the 19th Real-Time Systems Symposium RTSS'98*, IEEE Computer Society Press 1998, pages: 264–273.
- [4] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.
- [5] E. Allen Emerson and Joseph Y. Halpern. “Sometimes” and “Not Never” Revisited: On Branching versus Linear Time Temporal Logic. *Journal of the ACM* 33(1): Pages 151-178, 1986.
- [6] Y. Kesten, A. Pnueli, J Sifakis, and S. Yovine. Integration Graphs: A Class of Decidable Hybrid Systems. In *Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science*, pages 179-208. Springer Verlag, 1994.
- [7] Li Xuan Dong and Dang Van Hung. Checking Linear Duration Invariants by Linear Programming. In *Concurrency and Parallelism, Programming, Networking, and Security* LNCS 1179, pages. 321–332,1996.
- [8] Li Yong and Dang Van Hung. Checking Temporal Duration Properties of Timed Automata. In *Journal of Computer Science and Technology*, 17(6): 689-698 (2002).
- [9] Pham Hong Thai and Dang Van Hung. Verifying Linear Duration Constraints of Timed Automata. *ICTAC 2004*:295-309.
- [10] Zhao Jianhua and Dang Van Hung. Checking Timed Automata for Some Discretisable Duration Properties. In *Journal of Computer Science and Technology*, 15(5): 423-429 (2000).
- [11] Zhou Chaochen, C.A.R. Hoare, and Anders P. Ravn. A calculus of durations. *Information Processing Letters*, 40(5):269–276, 1991.
- [12] Chaochen Zhou. Linear Duration Invariants. *FTRTFT* 1994: 86-109.
- [13] Zhou Chaochen and Hansen M. R. *Duration Calculus. A Formal Approach to Real-Time Systems*, Springer Verlag, 2004.
- [14] Sergio Yovine: KRONOS: A Verification Tool for Real-Time Systems. *STTT* 1(1-2): 123-133 (1997).
- [15] G. Behrmann, A. David and K.G. Larsen. A tutorial on Uppaal. In *Formal Methods for the Design of Real-Time Systems (SFM-RT 2004)*, LNCS 3185, pages 200-236. Springer Verlag, 2004.

-
- [16] Paritosh K. Pandya. Interval Duration Logic: Expressiveness and Decidability. ENTCS 65(6), 2002.
 - [17] Meyer R, Faber J, Rybalchenko A. Model Checking Duration Calculus: A Practical Approach. LNCS 4281, Springer Verlag, 2007, pp 332–346.
 - [18] Fränzle M, Hansen M R. Deciding an Interval Logic with Accumulated Durations. LNCS 4424. Springer Verlag, 2007, pp 201–215.