

Interval Duration Logic: Expressiveness and Decidability

Paritosh K. Pandya^{1,2}

*School of Technology and Computer Science
Tata Institute of Fundamental Research
Homi Bhabha Road, Colaba,
Mumbai 400005, India*

Abstract

We investigate a variant of dense-time Duration Calculus which permits model checking using timed/hybrid automata. We define a variant of the Duration Calculus, called Interval Duration Logic, (*IDL*), whose models are timed state sequences [1].

A subset *LIDL* of *IDL* consisting only of *located time constraints* is presented. As our main result, we show that the models of an *LIDL* formula can be captured as timed state sequences accepted by an event-recording integrator automaton. A tool called *IDLVALID* for reducing *LIDL* formulae to integrator automata is briefly described. Finally, it is shown that *LIDL* has precisely the expressive power of event-recording integrator automata, and that a further subset *LIDL-* corresponds exactly to event-recording timed automata [2]. This gives us an automata-theoretic decision procedure for the satisfiability of *LIDL-* formulae.

1 Introduction

Duration Calculus (DC) [29,12] is a highly expressive logic for specifying quantitative timing properties of systems. It is closely related to the Interval Temporal Logic of Moszkowski [17]. It provides novel interval based modalities for describing behaviours.

For example, the following formula holds for a behaviour provided, in *any* time interval longer than 3 seconds where there is overload throughout, the alarm must be sounding at the end of the interval.

$$\Box([\textit{overload}] \wedge \ell > 3 \Rightarrow \textit{true} \frown [\textit{alarm}])$$

¹ Partially supported by the UNU/IIST offshore project *Semantics and verification of real-time programs using Duration Calculus: Theory and Practice*

² Email: pandya@tifr.res.in

Informally, this implies that once overload lasts for 3 seconds, the alarm must come on. Moreover, the alarm must then persist as long as overload lasts. In this formula, \square modality ranges over all time intervals within the behaviour. Each such time interval gives a fragment of the behaviour. The operator \frown is like concatenation (fusion) of behaviour fragments and $\llbracket \textit{overload} \rrbracket$ states invariance of *overload* over the behaviour fragment. Finally, ℓ measures the time length of the behaviour fragment (interval). Another kind of measurement is $\int P$ which measures the accumulated amount of time for which condition P holds within the interval. A precise definition of the syntax and semantics of Duration Calculus (variant) is given in Section 2.

Duration calculus (*DC*) was designed to be a convenient and highly expressive logic for specifying complex requirements over real-time systems [29]. Because of its high expressive power, Duration Calculus is also effective in formulation of compositional semantics of programming notations such as Esterel and Timed CSP [25,24].

In this paper, we consider the question of model checking Duration Calculus. Duration Calculus is a dense-time interval temporal logic whose models treat propositions as boolean functions of time (also called *signals* by some authors [4]). Unfortunately, the automata theory for Duration Calculus models (signals) is still evolving [4] and there are no available tools supporting such models. Another well-established model of real-time computations is the *timed state sequences* model [1]. Timed automata for recognising such timed state sequences are well investigated and there are now mature tools such as Hytech [3], Uppaal [5] and Kronos [6] for analysing these automata. Hence, for model checking, it seems preferable to work with timed state sequences rather than signals at present.

In this paper, we define a variant of the Duration Calculus, called Interval Duration Logic, (*IDL*), whose models are finite timed state sequences. It is a dense-time interval logic. *IDL* inherits much of the expressive convenience of original Duration Calculus, and most of the case studies using *DC* can be easily adapted to *IDL*. At the same time, there are significant technical differences between *DC* and *IDL*. The key advantage of *IDL* over *DC* is the availability of timed automata recognising *IDL* models. These automata can be used for satisfiability checking or model checking. Due to their high expressive power, the satisfiability of both *DC* and *IDL* turn out to be undecidable in general.

As our main contribution, we present a subset *LIDL* of *IDL* consisting only of *located time constraints*. As our main result, we show that the models of an *LIDL* formula can be captured as timed state sequences accepted by a finite state event-recording integrator automaton. We also show that *LIDL* has precisely the expressive power of event-recording integrator automata. Moreover, a subset *LIDL-* of *LIDL* exactly corresponds to the event-recording timed automata [2] whose emptiness is decidable [1]. This gives us an automata theoretic decision procedure for the satisfiability of *LIDL-*. Unlike

many decidable fragments of Duration Calculus, both *LIDL* and *LIDL-* are closed under all boolean operations including negation. A large number of examples of interest from Duration Calculus literature can be expressed within the subset *LIDL-*. Thus, we believe that *LIDL-* answers the quest for a dense-time Duration Calculus variant which is decidable and also practically interesting.

We have partially implemented a tool, called IDLVALID, which performs the reduction of *LIDL* formulae into event-recording integrator automata. A small example of its use is presented at the end of the paper. The resulting automata can be analysed using tools such as Hytech [3], Uppaal [5] and Kronos [6] to establish the satisfiability (validity) of *LIDL* formulae. Moreover, the generated automata can be used as observers for model checking *LIDL* properties [21].

The rest of the paper is organised as follows. The Interval Duration Logic is defined in Section 2. A minepump controller specification is given in Section 2.1 to illustrate the use of *IDL*. The subset *LIDL* is defined in Section 3, and the reduction of *LIDL* formulae to event-recording automata is given in Section 4. An example this of reduction, implemented using the tool IDLVALID, is given in Section 5. The paper ends with a discussion.

2 Interval Duration Logic

Let $Pvar$ be the set of propositional variables (called state variables in *DC*). The set of states is $\Sigma = 2^{Pvar}$ consisting of the set of subsets of $Pvar$. Let ω be the set of natural numbers $\{0, 1, \dots\}$.

Definition 2.1 A timed state sequence over $Pvar$ is a pair $\theta = (\sigma, \tau)$ where

- $\sigma = s_0 s_1 \dots s_{n-1}$ with $s_i \in 2^{Pvar}$ is a finite non-empty sequence of states, and
- $\tau = t_0 t_1 \dots t_{n-1}$ is a finite sequence of time stamps such that $t_i \in \mathbb{R}^0$ with $t_0 = 0$ and τ is non-decreasing.

Let $dom(\theta) = \{0, \dots, n-1\}$ be the set of positions within the sequence θ . Also, let the length of θ be $\#\theta = n$. A sequence θ' is called p -variant of θ if $\#\theta = \#\theta'$ and for all $q \neq p$, we have $q \in \theta'(i)$ **iff** $q \in \theta(i)$.

Here, it is assumed that the system evolves by discrete steps (transitions). Element s_i denotes the i 'th state and t_i gives the time at which this state is entered. Thus, the system remains in state s_i for the time interval $[t_i, t_{i+1})$ which includes time t_i but excludes time t_{i+1} . Note that in a timed state sequence there may be several positions with same time-stamp representing that several steps of computation take place at the same observable macro-time. In literature this is sometimes called super-dense computation (see [24] for its use). We shall also refer to timed state sequences as behaviours.

Let $Prop$ be the set of propositions over $Pvar$ with the following abstract

syntax. Let p range over $PVar$ and let P, Q range over $Prop$.

$$0 \mid 1 \mid p \mid P \wedge Q \mid \neg P \mid \ominus P$$

Here, 0 denotes proposition *false* and 1 represents *true*.

The truth of proposition P can be evaluated at any position i in $dom(\theta)$. Boolean combinators have their usual meaning. $\ominus P$ holds at a position i provided P holds at the previous position in the timed state sequence. We give a few clauses of this definition omitting the rest.

$$\begin{aligned} \theta, i \models p & \text{ iff } p \in s_i \\ \theta, i \models \ominus P & \text{ iff } i > 0 \text{ and } \theta, i-1 \models P \end{aligned}$$

Other boolean combinators \vee, \Rightarrow, \dots can be defined in the standard fashion. We also define some more operators.

$$\begin{aligned} \uparrow P & \stackrel{\text{def}}{=} (\ominus \neg P) \wedge P \\ \uparrow\uparrow P & \stackrel{\text{def}}{=} (\neg \ominus P) \wedge P \end{aligned}$$

Similarly, $\downarrow p, \downarrow\downarrow P$ can also be defined. Note that $\neg \ominus P$ is true at position 0 whereas $\ominus \neg P$ is false. At all other positions, they are identical.

Logic *IDL* is a form of interval temporal logic. The set of intervals within a timed state sequence θ can be defined as follows, where $[b, e]$ denotes a pair of positions.

$$Intv(\theta) = \{[b, e] \in dom(\theta)^2 \mid b \leq e\}$$

Each interval uniquely identifies a subsequence of θ .

Syntax of Interval Duration Logic

Let p, q range over propositional variables from $Pvar$, let P, Q range over propositions and D_1, D_2 range over *IDL* formulae. Let c_i range over integer constants and c_r range over rational constants.

$$\begin{aligned} [P]^0 \mid [P] \mid D_1 \frown D_2 \mid \overleftarrow{\diamond} D \mid D_1 \wedge D_2 \mid \neg D \mid \exists p. D \mid \\ \eta \text{ op } c_i \mid \Sigma P \text{ op } c_i \mid \ell \text{ op } c_r \mid \int P \text{ op } c_r \end{aligned}$$

where $op \in \{ < \mid > \mid = \mid \leq \mid \geq \}$.

Formulae of the form $\eta \text{ op } c_i$ or $\Sigma P \text{ op } c_i$ are called *discrete measurement formulae*, whereas formulae of the form $\ell \text{ op } c_r$ and $\int P \text{ op } c_r$ are called *dense measurement formulae*³.

³ We can easily extend measurement formulae to comparison of two terms involving arithmetic operators as in full Duration Calculus, e.g. $20 * \int P < \ell$.

Semantics of IDL

We define the truth of Interval Duration Logic formula D within a timed state sequence θ and interval $[b, e]$. This is denoted by $\theta, [b, e] \models D$.

$$\begin{aligned}
 \theta, [b, e] \models [P]^0 & \text{ iff } b = e \text{ and } \theta, b \models P \\
 \theta, [b, e] \models [P] & \text{ iff } b < e \text{ and for all } t : b < t < e. \theta, t \models P \\
 \theta, [b, e] \models D_1 \wedge D_2 & \text{ iff for some } m : b \leq m \leq e. \\
 & \theta, [b, m] \models D_1 \text{ and } \theta, [m, e] \models D_2 \\
 \theta, [b, e] \models \overleftarrow{\diamond} D & \text{ iff for some } b' \leq b : \theta, [b', e] \models D \\
 \theta, [b, e] \models D_1 \wedge D_2 & \text{ iff } \theta, [b, e] \models D_1 \text{ and } \theta, [b, e] \models D_2 \\
 \theta, [b, e] \models \neg D & \text{ iff } \theta, [b, e] \not\models D
 \end{aligned}$$

We also have existential quantification over propositional letters:

$$\begin{aligned}
 \theta, [b, e] \models \exists p. D & \text{ iff } \theta', [b, e] \models D \\
 & \text{ for some } \theta' \text{ which is } p\text{-variant of } \theta.
 \end{aligned}$$

Now we consider the semantics of measurement formulae. Logic *IDL* has four different types of measurement terms.

$$\eta \mid \Sigma P \mid \ell \mid \int P$$

These represent some specific quantitative measurements over the behaviour in a given interval. We shall denote the value of a measurement term t in a timed state sequence θ and an interval $[b, e]$ by $eval(t)(\theta, [b, e])$. This is defined below.

- Step Length η gives the number of steps within a given interval.

$$eval(\eta)(\theta, [b, e]) = e - b$$

- Time length ℓ gives the amount of real-time spanned by a given interval.

$$eval(\ell)(\theta, [b, e]) = t_e - t_b$$

- Step count ΣP counts the number of states for which P holds in the (right-closed-left-open) interval.

$$eval(\Sigma P)(\theta, [b, e]) = \sum_{i=b}^{e-1} \sigma(i)(P)$$

- Duration $\int P$ gives amount of real-time for which proposition P holds in given interval.

$$\begin{aligned}
 eval(\int P)(\theta, [b, e]) & = \\
 \sum_{i=b}^{e-1} & \left(\begin{array}{ll} t_{i+1} - t_i & \text{if } \sigma, i \models P \\ 0 & \sigma, i \not\models P \end{array} \right)
 \end{aligned}$$

Now we can define the semantics of measurement formula as follows

$$\theta, [b, e] \models t \text{ op } c \text{ iff } eval(t)(\theta, [b, e]) \text{ op } c$$

Finally, a formula D holds for a timed state sequence θ if it holds for the full interval spanning the whole sequence.

$$\begin{aligned} \theta \models D & \text{ iff } \theta, [0, \#\theta - 1] \models D \\ \models D & \text{ iff } \theta \models D \text{ for all } \theta \end{aligned}$$

Derived Operators

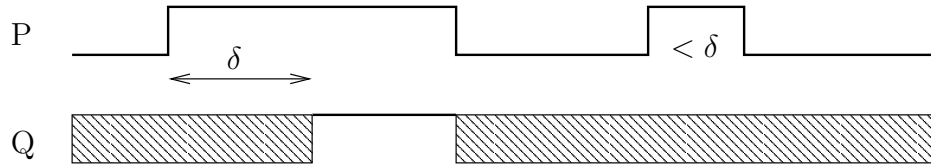
Note that $[1]^0$ holds for all point intervals whereas $[1]$ holds for all extended intervals. Formula $[P]^0 \wedge \text{true}$ states that P is true at the beginning of the interval. Now we define some derived operators.

- $\llbracket P \rrbracket \stackrel{\text{def}}{=} ([P]^0 \wedge [P] \wedge [P]^0)$ states that proposition P holds invariantly over the closed interval $[b, e]$ including the endpoints. Also, $\llbracket P \rrbracket \stackrel{\text{def}}{=} ([P]^0 \wedge [P])$
 $\llbracket P \rrbracket^+ \stackrel{\text{def}}{=} (\llbracket P \rrbracket \vee [P]^0)$. Similarly, $\llbracket P \rrbracket^+$ etc.
- $\text{unit} \stackrel{\text{def}}{=} [0]$ holds for intervals of the form $[b, b + 1]$.
- $\llbracket P \rrbracket^1 \stackrel{\text{def}}{=} [P]^0 \wedge \text{unit}$ holds for one step (two state) intervals where P is true at the beginning.
- $\diamond D \stackrel{\text{def}}{=} \text{true} \wedge D \wedge \text{true}$ holds provided D holds for some subinterval.
- $\square D \stackrel{\text{def}}{=} \neg \diamond \neg D$ holds provided D holds for all subintervals.
- We can specify the stability of a proposition as follows.

$$\text{stable}(P, \delta) \stackrel{\text{def}}{=} \square([\uparrow P]^0 \wedge (\ell < \delta) \Rightarrow \llbracket P \rrbracket^+)$$

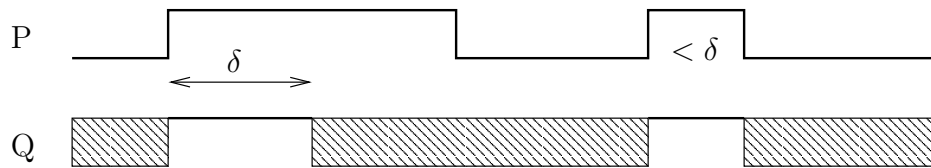
This states that once P becomes true, it must hold for δ time. Note that this also applies to P being true initially.

- $\llbracket P \rrbracket \xrightarrow{\delta} [Q]^0 \stackrel{\text{def}}{=} \neg \diamond (\llbracket P \rrbracket \wedge \ell \geq \delta \wedge \neg [Q]^0)$



This operator specifies that if P holds continuously for δ or more time, then Q must become true within the first δ time. Moreover, Q must then persist till P persists. Note that nothing is specified about the value of Q otherwise.

- $\llbracket P \rrbracket^+ \xrightarrow{\leq \delta} [Q]^0 \stackrel{\text{def}}{=} \neg \diamond ([\uparrow P]^0 \wedge (\llbracket P \rrbracket^+ \wedge \ell < \delta) \wedge \neg [Q]^0)$



This operator specifies that once P becomes true, for the first δ time, while P persists Q will also be true.

The last three operators are adapted from Ravn [27] where a systematic methodology for specifying and verifying real-time systems has been expounded. It is possible to define many subtle variants of these operators in *IDL*. Note that operator $\overleftarrow{\Diamond} D$ provides a backward expanding modality not usually considered in *DC*. This operator, together with a forward expanding modality, was first investigated by Halpern and Shoham [9]. We will make a significant use of it later in the paper.

2.1 Mine pump

A mine has water seepage which must be removed by operating a pump. There is a high water sensor. In response to water being high, a pump may be operated. The pump must not operate if the water is not high. The mine also has pockets of methane which escape. The presence of methane is detected by a sensor. When there is methane, all electrical activity including the pump must be shut down to prevent explosion.

We shall present a model the mine pump system in *IDL* and establish that under suitable assumptions the water level never becomes dangerous.

- HH_2O Water level is high.
- DH_2O Water level is dangerous.
- HCH_4 Methane level is high.
- $PumpOn$ The pump is operating.

Water Seepage Assumptions

High water level occurs before Danger water level. (Sensor is reliable.)

$$As_1 \stackrel{\text{def}}{=} \Box(\llbracket DH_2O \Rightarrow HH_2O \rrbracket^+)$$

It takes at least w seconds for the water level to turn dangerous after reaching the high water level

$$As_2 \stackrel{\text{def}}{=} \llbracket HH_2O \rrbracket^+ \overset{<w}{\leftrightarrow} \llbracket \neg DH_2O \rrbracket^0$$

This property specifies a minimum separation of w time between water level becoming high and its becoming dangerous. The value of w must be calculated based on estimates of rate of water seepage.

Pump control

Pump enabling condition: It is safe to operate the pump only when water is high and there is no methane.

$$SafePump \stackrel{\text{def}}{=} HH_2O \wedge \neg HCH_4$$

Pump is started within δ seconds of it being enabled. The pump will remain on while it is enabled.

$$Pc_1 \stackrel{\text{def}}{=} \llbracket SafePump \rrbracket \overset{\delta}{\rightarrow} \llbracket PumpOn \rrbracket^0$$

Pump is stopped within δ seconds of being disabled.

$$Pc_2 \stackrel{\text{def}}{=} [[\neg SafePump] \xrightarrow{\delta} [\neg PumpOn]]^0$$

The conjunction of above two properties is called *Pumpcontrol*

Pump Capacity Assumption

Pump can bring water level down below high water level within ϵ seconds (from any starting condition).

$$As_3 \stackrel{\text{def}}{=} [[PumpOn] \xrightarrow{\epsilon} [\neg HH_2O]]^0$$

Methane Release Assumptions

Between two occurrences of Methane Release there is at least ζ seconds.

$$As_4 \stackrel{\text{def}}{=} \Box([\downarrow HCH_4]^0 \wedge [[\neg HCH_4] \wedge [HCH_4]]^0 \\ \Rightarrow \ell > \zeta)$$

The high methane level lasts at most κ time units.

$$As_5 \stackrel{\text{def}}{=} \Box([\uparrow HCH_4] \Rightarrow \ell < \kappa)$$

$$As_6 \stackrel{\text{def}}{=} (2 * (\delta + \epsilon) + \kappa) < w < \zeta$$

Verification Condition

$$(As_1 \wedge As_2 \wedge As_3 \wedge As_4 \wedge As_5 \wedge As_6 \wedge PumpControl) \\ \Rightarrow (\Sigma DH_2O = 0)$$

The reader is urged to convince himself that the above verification condition is indeed valid. Liu Zhiming [15] has established correctness of a somewhat similar specification of Mine pump using the proof rules of Duration calculus. In Section 4 we shall discuss how this condition can be automatically verified using a validity checking procedure for some given values of constants $\delta, w, \epsilon, \kappa, \zeta$.

2.2 Decidability

Theorem 2.2 *The satisfiability of IDL formulae is undecidable* □

The proof is a straightforward variant of the undecidability proof for Duration Calculus [30], where for every 2-counter machine a formula can be constructed such that the every model of the formula defines a halting run of the 2-counter machine. We omit the proof here.

Sub-logic QDDC

Consider the subset of *IDL* where dense-time measurement constructs of the form $\ell \text{ op } c_r$ or $\int P \text{ op } c_r$ are not used. This subset is called Quantified Discrete-time Duration Calculus, (*QDDC*). Note that discrete time measurement constructs $\eta \text{ op } c_i$ or $\Sigma P \text{ op } c_i$ can still be used.

Theorem 2.3 *For every QDDC formula D over variables $Pvar$, we can effectively construct a finite state automaton $A(D)$ over the alphabet 2^{Pvar} such that for all state sequences $\sigma \in (2^{Pvar})^*$,*

$$\sigma \models D \quad \text{iff} \quad \sigma \in L(A(D))$$

Hence, satisfiability of QDDC formulae is decidable. □

A separate paper gives details of logic QDDC and its automata theoretic decision procedure [20]. We omit these proofs here. A tool called DCVALID has been implemented for model checking of QDDC formula [19,20]. As the tool constructs the automaton representing models of a formula, this can be used to visualise models and counter models, and also for model checking the specification against designs [21,23].

It should be mentioned that the lower bound on the size of automaton $A(D)$, in the worst case, is non-elementary. However, such blowup is rarely observed in practice and we have been able to check validity of many reasonably sized formulae with our tool DCVALID [19,20,23]. DCVALID is implemented using MONA [13], which is a sophisticated tool for deciding logic WS1S.

3 Located Constraints

We now consider the question of finding a reasonable subset of *IDL* with dense measurements which is decidable. The key idea is that we restrict the use dense measurement formulae to *Located Constraints*.

A located constraint has the form $P \rightsquigarrow M$ where P is a proposition and M is a dense time constraint of the form $\ell \text{ op } c_r$ or $\int Q \text{ op } c_r$. Proposition P is called anchor and M is called the constraint. Such a formula is read as *since last P , constraint M holds*.

Definition 3.1 Semantics of located constraints:

$$(P \rightsquigarrow M) \stackrel{\text{def}}{=} [1]^0 \wedge \overleftarrow{\diamond} (M \wedge ([P]^0 \wedge [\neg P]))$$

Proposition 3.2 *Semantics of located constraints.*

$$\begin{aligned} \theta, [b, e] \models (P \rightsquigarrow M) \quad & \text{iff} \\ b = e \quad \text{and} \quad \exists b' < b. \theta, b' \models P \quad & \text{and} \\ \forall b' < b'' < e. \theta, b'' \not\models P \quad \text{and} \quad \theta, [b', e] \models M \end{aligned}$$

Interval Duration Logic with Located Constraints (LIDL)

The subset of *IDL* where dense-time properties occur only as located constraints is called *IDL* with located constraints (*LIDL*).

The syntax of logic *LIDL* is defined below. Let

$$D ::= [P]^0 \mid \lceil P \rceil \mid D_1 \frown D_2 \mid \overleftarrow{\diamond} D \mid D_1 \wedge D_2 \mid \neg D \mid \exists p.D \mid \\ \eta \text{ op } c_i \mid \Sigma P \text{ op } c_i \mid (P \rightsquigarrow M)$$

$$M ::= \ell \text{ op } c_r \mid \int Q \text{ op } c_r$$

$$\text{op} \in < \mid > \mid = \mid \leq \mid \geq$$

The syntactic restriction is that located constraint sub-formula ($P \rightsquigarrow M$) does not occur in scope of any $\exists p$ if $p \in \text{var}((P \rightsquigarrow M))$.

Let *LIDL*– denote the set of *LIDL* formulae which do not use any term of the form $\int P$. Note that term ℓ can still be used as shown in examples below.

Examples

Logic *LIDL*– is expressive enough to state many of the timing requirements we have encountered in Duration Calculus case studies. These requirements typically have the form of formulae given in the minepump example (Section 2.1). We show how these can be formulated in the sub-logic *LIDL*–. All the formulae mentioned below refer to the Minepump specification of Section 2.1.

- Consider the formula As_4 of Section 2.1. This formula states a minimum time separation between two events. In *LIDL* it can be stated as:

$$\square([\downarrow HCH_4]^0 \frown [\lceil \neg HCH_4 \rceil] \frown [HCH_4]^0 \\ \Rightarrow \text{true} \frown (\downarrow HCH_4 \rightsquigarrow \ell \geq \zeta))$$

- Consider the formula As_5 in Section 2.1. This formula puts upper bound on time of existence of condition HCH_4 . In *LIDL* this can be stated as:

$$\square([\lceil HCH_4 \rceil] \rightarrow \text{true} \frown (\uparrow HCH_4 \rightsquigarrow \ell \leq \kappa))$$

- The first formula Pc_1 of pump control specification can be stated as:

$$\square([\lceil \text{SafePump} \rceil] \frown (\uparrow \text{Safepump} \rightsquigarrow \ell \geq \delta)) \Rightarrow \text{true} \frown [\text{PumpOn}]^0$$

- The water seepage assumption As_2 can be formulated as:

$$\square([\lceil HH_2O \rceil]^+ \frown (\uparrow HH_2O \rightsquigarrow \ell < w)) \Rightarrow \text{true} \frown [\lceil \neg DH_2O \rceil]^0$$

The last two formulae are examples of “arrow” operators of Ravn [27]. All the formulae of Minepump example can be expressed in *LIDL*– in this fashion.

4 Decidability of $LIDL-$

We first introduce the notion of event recording integrator automata before considering decidability.

Integrator Automata

These are a sub-class of hybrid automata where only clocks (whose rate is always 1) or integrators (whose rate is either 0 or 1) are used [14].

An *Integrator Automaton* AT is the tuple $(Q, \Sigma, q_0, F, C, \delta)$. It has a finite set of states Q with initial state q_0 , an alphabet Σ , a set of final states F , a set of integrators (clocks) C and a transition relation δ . The transition relation has the type

$$\delta \subseteq Q \times \Sigma \times 2^C \times \Phi_C \times [C \rightarrow \{0, 1\}] \times Q$$

where each transition $(q, a, X, \phi, \gamma, q')$ is guarded by a condition $\phi \in \Phi_C$ and it resets a subset of integrators X . The condition ϕ is a conjunction of constraints of the form $x \text{ op } r$ where $x \in C$ is a integrator and r is a rational constant. The integrator rate assignment $\gamma : C \rightarrow \{0, 1\}$ assigns to each integrator a rate 0 or 1.

An *integrator valuation* $\nu : C \rightarrow \mathbb{R}$ assigns to each integrator a real (time) value. $\nu + \gamma \cdot t$ increments those integrators whose rate is 1 by amount t , i.e. $(\nu + \gamma \cdot t)(c) = \nu(c) + \gamma(c) \cdot t$, and $\nu \oplus [X \mapsto 0]$ replaces the value of each integrator in X to 0, i.e. $(\nu \oplus [X \mapsto 0])(c)$ is 0 if $c \in X$ and $\nu(c)$ otherwise.

A run of AT is a finite sequence of the form

$$(q_0, \nu_0, e_0) \xrightarrow{a_0, X_0, \gamma_0} (q_1, \nu_1, e_1) \dots (q_{n-1}, \nu_{n-1}, e_{n-1}) \xrightarrow{a_{n-1}, X_{n-1}, \gamma_{n-1}} \dots (q_n, \nu_n, e_n)$$

where (a) q_0 is the initial state, (b) $\nu_0(x) = 0$ for each $x \in C$, (c) $e_i \in \mathbb{R}$ gives the amount of real-time time spent in state q_i , with $e_0 = 0$, (d) for each $i > 0$, let the valuation $\nu'_i = \nu_i + \gamma_{i-1} \cdot e_i$, and $\nu'_0(x) = 0$ for all x . Let $\nu_{i+1} = \nu'_i \oplus [X_i \mapsto 0]$. Then, for each i , we must have some $\phi_i \in \Phi_C$ such that $(q_i, a_i, X_i, \phi_i, \gamma_i, q_{i+1}) \in \delta$ and $\nu'_i \models \phi_i$.

The timed state sequence of the run is (σ, τ) with $\sigma = (a_0, a_1, \dots, a_{n-1})$ and $\tau = (t_0, t_1, \dots, t_{n-1})$. Here $t_0 = 0$ and $t_{i+1} = t_i + e_{i+1}$.

The run is accepting if the last state $q_n \in F$. The language of the automaton, $L(AT)$, is the set of timed state sequences arising from all accepting runs of AT .

In an **event recording integrator automaton**, the resetting of integrators is determined by the alphabet associated with the transition, and the integrator rates are also determined by the alphabet. Thus, there are functions

$$reset : \Sigma \rightarrow 2^C$$

$$rate : \Sigma \rightarrow [C \rightarrow \{0, 1\}]$$

such that for any transition $(q, a, X, \phi, \gamma, q') \in \delta$, we have $X = reset(a)$ and $\gamma = rate(a)$.

An integrator automaton is called timed automaton if for all transitions and for all integrators x , we have $\gamma(x) = 1$, i.e. integrator rates are always 1. Such integrators are called *clocks*. An event recording integrator automaton meeting above condition is called *event recording timed automaton*. This is an important sub-class of timed automata which can be determinized and also complemented [2].

Decidability

We follow an automata theoretic approach to deciding *LIDL* formulae.

Theorem 4.1 *For every $D \in \text{LIDL}$ we can effectively construct an event recording integrator automaton $A(D)$ such that*

$$\theta \models D \quad \text{iff} \quad \theta \in L(A(D))$$

Moreover, if $D \in \text{LIDL-}$, i.e. it is free of terms $\int P$, then $A(D)$ is an event recording timed automaton. \square

We shall outline the proof of this main theorem later in this section.

Corollary 4.2 *Satisfiability of any LIDL- is decidable.*

Proof. By Theorem 4.1, for any formula D of LIDL- , we can construct an event-recording timed automaton $A(D)$ accepting the models of D . Since the emptiness of timed automata in general is decidable [1], we can determine whether D has any models. \square

Theorem 4.3 *For every event recording integrator automaton A we can construct a formula $D(A) \in \text{LIDL}$ such that*

$$\theta \models D(A) \quad \text{iff} \quad \theta \in L(A)$$

Moreover, if A is an event recording timed automaton (i.e. uses only clocks), then $D(A) \in \text{LIDL-}$, i.e. it does not use terms of the form $\int P$. \square

We omit the proof of this theorem. It can be found in the full version of the paper. An encoding of accepting runs of Timed Buchi Automaton into Duration Calculus was given in [18]. A similar encoding can be used here.

Thus, *LIDL* has exactly the expressive power of event recording integrator automata, and *LIDL-* has exactly the expressive power of event recording timed automata.

4.1 Proof of Theorem 4.1

We now give an outline of the proof of our main theorem, Theorem 4.1. A detailed proof will be available in the full version of this paper.

Our aim is to construct an integrator automaton $A(D)$ for a given *LIDL* formula D . Let the formula be $D(\phi_1, \dots, \phi_k)$ where ϕ_i denotes an occurrence of a located constraint $P_i \rightsquigarrow M_i$. Let $\text{var}(D) = \text{Pvar}$. Let $\text{TW}(\text{Pvar})$ denote the set of all timed state sequences over Pvar . We shall give the construction of $A(D)$ in a series of steps.

Step 1

Let $Lvar = L_1, \dots, L_k$ be fresh prop. variables. We will use L_i as a witness for ϕ_i .

Definition 4.4 Let $\theta' \in TW(Pvar \cup Lvar)$. Define θ' to be **consistent** if $\#\theta' = \#\theta$ and $\forall i \in dom(\theta'), \theta', i \models L_i$ **iff** $\theta, i \models \phi_i$.

Let $CONSIST$ be set of consistent timed state sequences. For $\theta \in TW(Pvar)$, let $\hat{E}(\theta) \in TW(Pvar \cup Lvar)$ denote the unique consistent extension of θ .

Step 2

Define the witness formula as follows.

$$witness(P, L) \stackrel{\text{def}}{=} [L]^0 \wedge \overline{\diamond} ([P]^0 \wedge [\neg P])$$

Let $E(D) \stackrel{\text{def}}{=} D[witness(P_i, L_i) \wedge \phi_i / \phi_i]$

$U(D) \stackrel{\text{def}}{=} D[witness(P_i, L_i) / \phi_i]$

Claim 4.5 For all $I' \in CONSIST$,

$$\theta' \models D \quad \text{iff} \quad \theta' \models E(D) \quad \text{iff} \quad \theta' \models U(D)$$

Step 3

Let $U(D)$ be as above. Note that $U(D) \in QDDC$. By Theorem 2.3, we can construct a total and deterministic automaton $A(U(D))$ accepting exactly the models of $U(D)$. Denote by $\hat{L}(A(U(D)))$ the set of timed state sequences over $Pvar \cup Lvar$ whose untimed parts are in $L(A(U(D)))$, the language of words accepted by $A(U(D))$.

Claim 4.6 For all $\theta' \in CONSIST$, $\theta' \models U(D)$ **iff** $\theta' \in \hat{L}(A(U(D)))$

Step 4

We construct an Integrator Automaton A' from $A(U(D))$ as follows.

- For each ϕ_i we introduce an x_i .
If $\phi_i = P_i \rightsquigarrow \ell \text{ op } c_i$ then x_i is a clock.
If $\phi_i = P_i \rightsquigarrow \int Q_i \text{ op } c_i$ then x_i is an integrator.
- Let $Q(A') = Q(A(U(D)))$, with the same initial and final states as $A(U(D))$
For every transition $s \xrightarrow{v} s'$ in $A(U(D))$, we define a corresponding transition $s \xrightarrow{v, \psi} s'$ in A' , where test $\psi \in \Phi_C$ is given below. Recall that $v \in (Pvar \cup Lvar \rightarrow \{0, 1\})$. Test ψ is the conjunction of set of clock constraints

$$\{x_i \text{ op}_i c_i \mid v \models L_i\} \cup \{\neg(x_i \text{ op}_i c_i) \mid v \not\models L_i\}$$

where ϕ_i is $P_i \rightsquigarrow \ell \text{ op}_i c_i$ or $P_i \rightsquigarrow \int Q_i \text{ op}_i c_i$.

- The *reset* and *rate* functions are defined as follows. Recall that integrator x_j is introduced for a located constraint ϕ_j .

$$\begin{aligned} \text{rate}(v)(x_j) &= 1 && \text{if } x_j \text{ is a clock} \\ &1 && \text{if } \phi_j = (P_j \rightsquigarrow \int Q_j \text{ op } c_j) \text{ and } v \models Q_j \\ &0 && \text{otherwise} \end{aligned}$$

$$\text{reset}(v) = \{x_j \mid \phi_j = (P_j \rightsquigarrow M_j) \text{ and } v \models P_j\}$$

This completes the construction of A' .

Claim 4.7 *For all $\theta' \in \text{CONSIST}$, we have $\theta' \in L(A(U(D)))$ iff $\theta' \in L(A')$.*

Claim 4.8 *If $\theta' \in L(A')$ then $\theta' \in \text{CONSIST}$.*

Step 5

Note that $L(A')$ has alphabet $2^{Pvar \cup Lvar}$. By projecting the alphabet to 2^{Pvar} , we get the desired automaton A'' s.t.

$$\theta \models D \quad \text{iff} \quad \theta \in L(A'') \quad \square$$

5 Tool IDLVALID

We illustrate the construction of an integrator automaton from an *LIDL* formula by an example. Consider the formula

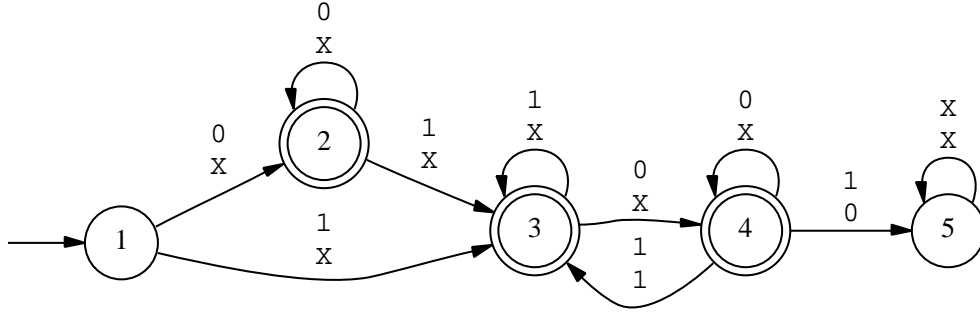
$$\begin{aligned} &\square([\downarrow HCH_4]^0 \frown [\neg HCH_4] \frown [HCH_4]^0) \\ &\Rightarrow \text{true} \frown (\downarrow HCH_4 \rightsquigarrow \ell \geq \zeta) \end{aligned}$$

First, we must replace the located constraints by witness formulae as in Step 2 of the construction (Section 4.1). We obtain a pure QDDC formula. In the above example, we introduce *witness*($\downarrow HCH_4, L$) in place of ($\downarrow HCH_4 \rightsquigarrow \ell \geq \zeta$).

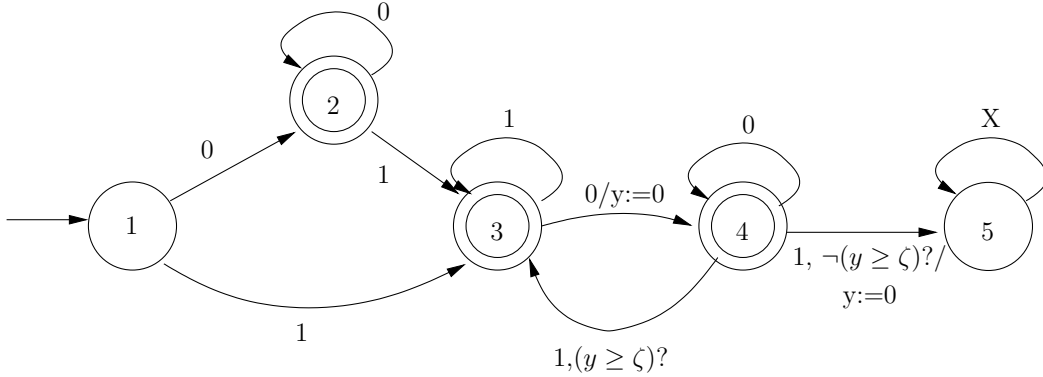
$$\begin{aligned} &\square([\downarrow HCH_4]^0 \frown [\neg HCH_4] \frown [HCH_4]^0) \\ &\Rightarrow \text{true} \frown ([L]^0 \wedge \overleftrightarrow{\diamond} ([\downarrow HCH_4]^0 \frown [\neg \downarrow HCH_4])) \end{aligned}$$

Next, an (untimed) automaton must be constructed for this formula as in Step 3 of the construction. (See Theorem 2.3 and following remarks.) We have implemented a tool called DCVALID to do this. A separate report describe this tool in detail [20,19]. It returns the following untimed automaton, were the alphabet of the automaton is a column vector of two bits denoting the values of HCH_4 and witness L . Value X denotes don't care (i.e. either 0 or 1 gives the same result). Note that our models are non-empty finite sequences. Hence initial state is not an accepting state. Also note that the automaton is

total and deterministic.



Finally, the generated untimed automaton must be transformed to an event recording integrator automaton, as in Steps 4 and 5 of the construction. In our example, as there is only one located constraint, we introduce a single clock y which must be reset whenever $\downarrow HCH_4$ holds. Also, a test $y \geq \zeta$ must be performed wherever $L = 1$ and test $\neg(y \geq \zeta)$ must be performed wherever $L = 0$. The transformed automaton is shown below. We have labelled each edge with value of HCH_4 and also shown the tests and resets.



In this automaton, timed state sequences which leads to states 2,3 or 4 are models of the formula. Timed state sequences which lead to state 5 are counter-models. Note that the resulting automaton is total and deterministic.

A tool called IDLVALID which fully automates this construction is currently under development. As mentioned, parts of it for generating untimed automata from QDDC formulae are already automated.

Once a hybrid/timed automaton accepting models of an *LIDL* formula D is constructed, the automaton can be analysed. Existing model checking tools such as Hytech [3], Uppaal [5] or Kronos [6] can be used to find if a final state is reachable in the constructed automaton. Such a reachability establishes the satisfiability of the original formula. Moreover, the constructed automaton can also be run as a synchronous observer in parallel with a system for model checking [21].

6 Discussion

The connection between logics and automata has greatly influenced the applicability of formal methods using model checking tools. In this paper, we have investigated a Duration Calculus like logic which can be model checked using timed/hybrid automata. Duration calculus (*DC*) is a well-established notation for specifying complex properties of real-time systems, with numerous case studies illustrating its high expressive power. However, it has lacked effective tool support till now.

In this paper, we have defined a variant of *DC*, called *IDL*, whose models are timed state sequences. We have identified a subset of *IDL*, called *LIDL*, whose formulae can be translated to event-recording integrator automata which precisely recognise the models of the formula. In fact, *LIDL* has exactly the expressive power of event-recording integrator automata and its further subset *LIDL*– has exactly the power of event-recording timed automata. Both these subsets are closed under boolean operations including negation.

The reduction from *LIDL* to timed/hybrid automata is practically important as such automata can be analysed using well-established tools such as Hytech, Uppaal and Kronos. Moreover, the automata can be used as property observers [21] for model checking. We have partially implemented the translation of *LIDL* formulae to integrator automata into a tool called IDLVALID (see Section 5).

Typical real-time requirements which arise in many *DC* case studies can be expressed within *LIDL*– showing that it is a practically interesting subset. These include boolean combinations of properties such as minimum time separation between two conditions, upper bound on how long a condition can last as well as the “arrow” operators of Ravn [27] (see the examples at the end of Section 3). Thus, we believe that *LIDL*– answers the long standing quest for a Duration Calculus like logic which is decidable and which can be model checked.

One natural question is whether the change from original Duration Calculus to *IDL* is really necessary for model checking. The original Duration Calculus formulae have the desirable property of being closed under stuttering which is not the case for *IDL*. In spite of this we have used *IDL* as its models are supported by a well-established theory of timed/hybrid automata which can be analysed using existing tools. It should be noted that the automata theory of original Duration Calculus models (signals [4]) is still evolving and we are not aware of any tools for them. At present, we are investigating a subset of *IDL* formulae which are stutter-closed. For these formulae, the *DC* and the *IDL* semantics will coincide. We believe that most natural *DC* specifications, including the specification of mine-pump example, have this property. Thus, the differences between *DC* and *IDL* are less important in practice and many of the *DC* case studies can be easily adapted to *IDL*.

Related Work

There has been considerable interest in finding subsets of dense time Duration Calculus which are decidable and model checkable. Some early results appeared in [30]. A class of Duration Calculus formulae called Linear Duration Invariants has been shown to be decidable by Zhou *et al* [31]. However, these formulae seem quite different from the properties which can be expressed in *LIDL*. Also notable are the results of Boujjani *et al* [7] and Franzle [10] on decidable subsets of Duration Calculus. Unlike our *LIDL* and *LIDL-*, both these subsets are not closed under boolean operations (esp. negation). In another approach, Dierks *et al* translate some important *DC* formulae into PLC-automata and then translate these into timed automata [8]. Again, the closure under boolean operations does not hold. The fact that *LIDL* can express many Duration Calculus case studies makes it significant. For discrete time duration calculus, a sub-logic QDDC has been shown to be decidable [20] and based on this a tool called DCVALID has been constructed for model checking SMV, Esterel and Verilog designs [21,22,23].

Some other decidable dense-time temporal logics include the monadic logic of relative distance due to Wilke [28] and the Event Clock Logic (*ECL*) of Raskin *et al* [26]. In comparison, *LIDL* is an *interval temporal logic* providing a qualitatively different vocabulary for expressing complex properties (see [18,20]). One consequence of this is that expressing some *LIDL-* formulae in *ECL* can necessarily result in a non-elementary blow-up in the size of the formula. *ECL* can express general liveness properties (e.g. infinitely often P) where as *IDL* can only express bounded liveness properties. Another difference is that *LIDL-* is expressively complete for event recording timed automata (see Theorem 4.3).

References

- [1] Alur, R., and D.L. Dill: Automata for modeling real-time systems, in *Proc. of 17th ICALP*, LNCS 443, Springer-Verlag, 1990.
- [2] Alur, R., L. Fix and T.A. Henzinger, A determinizable class of timed automata. In *Proc. 6th Conference on Computer-Aided Verification*, LNCS 818, Springer-Verlag, 1994.
- [3] Alur, R., T.A. Henzinger and P.H. Ho, Automatic Symbolic Verification of Embedded Systems, in *IEEE Transactions on Software Engineering*, **22**:181-201, 1996.
- [4] Asarin, E., P. Caspi and O. Maler, A Kleene theorem for timed automata. In *Proc. 12th Annual IEEE Symposium on Logic in Computer Science (LICS'97)*, Warsaw, IEEE Computer Society, 1997.
- [5] Bengtsson, Johan, K.G. Larsen, F. Larsson, P. Pettersson and Wang Yi, Uppaal — a Tool Suite for Automatic Verification of Real-Time Systems, in *Proc.*

- of Workshop on Verification and Control of Hybrid Systems III, LNCS 1066, Springer-Verlag, 1995.
- [6] Bozga, M., C. Daws, O. Maler, A. Olivero, S. Tripakis and S. Yovine, Kronos: A Model Checking Tool for Real-Time Systems, *Proc. 10th Int. Conf. on Computer Aided Verification*, LNCS1427, Springer-Verlag, 1998.
 - [7] Bouajjani, A., Y. Lakhnech and R. Robbana, From Duration Calculus to Linear Hybrid Automata, Proceedings of the 7th International Conference On Computer Aided Verification, CAV'95, LNCS 939, Springer-Verlag, 1995.
 - [8] Dierks, H., A. Fehnker, A. Mader and F. Vaandrager, Operational and Logical Semantics for Polling Real-Time Systems, In *Proc. FTRTFT'98*, Lyngby, Denmark, (eds.) A.P.Ravn and H. Rischel, LNCS 1486, Springer-Verlag, 1998.
 - [9] Halpern, J., and Y. Shoham, A propositional modal logic of time intervals, *JACM*, 38(4), 1991.
 - [10] Franzle, M., Model Checking Dense-Time Duration Calculus, In *Duration Calculus: A Logical Approach to Real-Time*, Systems Workshop proceedings of the 10th European Summer School in Logic, Languages and Information (ESSLLI X), Saarbrücken, Germany, 1998.
 - [11] Hansen, M.R., Model-Checking Duration Calculus, *Formal Aspects of Computing*, 1994.
 - [12] Hansen, M.R., and Zhou Chaochen, Duration Calculus: Logical Foundations, *Journal of Formal Aspects of Computing* **9**, 1997.
 - [13] Klarlund, N., A. Møller and M.I. Schwartzbach, MONA implementation secrets, to appear in *Proc. CIAA 2000*, 2000.
 - [14] Kesten, Y., A. Pnueli, J. Sifakis and S. Yovine, Integration graphs: A Class of Decidable Hybrid Systems, in *Proc. Hybrid Systems*, LNCS 736, Springer-Verlag, 1993.
 - [15] Liu, Z., Specification and verification in the duration calculus, in M. Joseph (ed.), *Real-time Systems: Specification, Verification and Analysis*, Prentice Hall, 1996.
 - [16] Liu, Z., A.P. Ravn, and X. Li, Compositional inductive verification of duration properties of real-time systems., In *Proc. of PROCOMET'98*, (eds.) Gries, D. and W.P. deRoever, Shelter Island, New York, Chapman and Hall, 1998.
 - [17] Moszkowski, B., A Temporal Logic for Multi-Level Reasoning about Hardware, in *IEEE Computer*, **18**(2), 1985.
 - [18] Pandya, P.K., Weak chop inverses and liveness in mean-value calculus, in *proc. FTRTFT'96*, Uppsala, Sweden, LNCS 1135, Springer-Verlag, 1996.
 - [19] Pandya, P.K., DCVALID User Manual, Tata Institute of Fundamental Research, Bombay, 1997. (Available in revised version at <http://www.tcs.tifr.res.in/~pandya/dcvalid.html>).

- [20] Pandya, P.K., Specifying and Deciding Quantified Discrete-time Duration Calculus Formulae using DCVALID: An Automata Theoretic Approach, In *Proc. Workshop on Real-time Tools (RTTOOLS'2001)*, Aalborg, Denmark, August 2001.
- [21] Pandya, P.K., Model checking CTL*[DC], In *Proc. TACAS 2001*, Genova, Italy, LNCS 2031, Springer-Verlag, 2001.
- [22] Pandya, P.K., Model checking CTL[DC] specifications of SMV, Verilog and Esterel Designs, Technical Report, TCS-00-PKP-3, Tata Institute of Fundamental Research, September 2000.
- [23] Pandya, P.K., The saga of synchronous arbiter: On model checking quantitative timing properties of synchronous programs, in *Proc. Workshop on Synchronous languages, applications and programming, SLAP'2002*, Grenoble, France (affiliated with EATPS'2002). *Electronic Notes in Theoretical Computer Science*, ENTCS 65.5, Elsevier Science B.V., April 2002.
- [24] Pandya, P.K., and H.V. Dang, A Duration Calculus of Weakly Monotonic Time, In *Proc. FTRTFT'98*, Lyngby, Denmark, (eds.) A.P.Ravn and H. Rischel, LNCS 1486, Springer-Verlag, 1998.
- [25] Pandya, P.K., Y.S. Ramakrishna, R.K. Shyamasundar. A Compositional Semantics of Esterel in Duration Calculus. In *Proc. Second AMAST workshop on Real-time Systems: Models and Proofs*, Bordeaux, June, 1995.
- [26] Raskin, J., and P. Schobben, State clock logic: a decidable real-time logic, In *Proc. HART'97*, LNCS 1021, Springer-Verlag, 1997.
- [27] Ravn, A.P., Design of Real-time Embedded Computing Systems, Department of Computer Science, Technical University of Denmark, 1994.
- [28] Wilke, T., Specifying Timed State Sequences in Powerful Decidable Logics and Timed Automata, *Proc. FTRTFT'94*, LNCS 863, Springer-Verlag, 1994.
- [29] Zhou, Chaochen, C.A.R. Hoare and A.P. Ravn, A Calculus of Durations, *Info. Proc. Letters*, **40**(5), 1991.
- [30] Zhou, Chaochen, M.R. Hansen and P. Sestoft, Decidability and Undecidability Results for Duration Calculus, In *STACS'93*, LNCS 665, Springer-Verlag, 1993.
- [31] Zhou, Chaochen, Zhang Zingzhou, Yang Lu and Li Xiaoshan, Linear Duration Invariants, in *Proc. FTRTFT'94*, LNCS 863, Springer-Verlag, 1994.