Bounded Validity Checking of Interval Duration Logic

Babita Sharma[1], Paritosh. K. Pandya[2], and Supratik Chakraborty[1]

[1] Indian Institute of Technology, Bombay, India
babita@cfdvs.iitb.ac.in, supratik@cse.iitb.ac.in
[2] Tata Institute of Fundamental Research, India
pandya@tifr.res.in

Abstract. A rich dense-time logic, called Interval Duration Logic (IDL), is useful for specifying quantitative properties of timed systems. The logic is undecidable in general. However, several approaches can be used for checking validity (and model checking) of IDL formulae in practice. In this paper, we propose bounded validity checking of IDL formulae by polynomially reducing this to checking unsatisfiability of *lin-sat* formulae. We implement this technique and give performance results obtained by checking the unsatisfiability of the resulting *lin-sat* formulae using the ICS solver. We also perform experimental comparisons of several approaches for checking validity of IDL formulae, including (a) digitization followed by automata-theoretic analysis, (b) digitization followed by pure propositional SAT solving, and (c) *lin-sat* solving as proposed in this paper. Our experiments use a rich set of examples drawn from the Duration Calculus literature.

1 Introduction

Interval Duration Logic (IDL)[16] is a highly expressive logic for specifying properties of real-time systems. It is a variant of Duration Calculus (DC) [20] with finite *timed state sequences* as its models. IDL, like DC, is a dense-time interval temporal logic, incorporating the notion of cumulative amount of time (duration) for which a condition holds in a given time interval. Because of this, IDL is well-suited for describing complex properties of real-time systems, including scheduling and planning constraints. A large number of case studies have exemplified the expressive power of the logic [19].

It has been shown that because of its rich expressive power, the problem of satisfiability (validity) checking of IDL (and DC) formulae is undecidable [16]. In spite of this, for reasons of practical applicability, there has been interest in developing tools and techniques for validity and model checking of various duration logics [5, 8, 16]. One approach has been to work with discrete-time versions of duration calculus [14, 9, 18]. More recently, a digitization approach which approximates dense-time validity of IDL by discrete-time validity has been proposed [6]. But model checking of dense-time interval logics like DC and IDL remains a challenging problem. So far, there have been no tools available for model checking

N. Halbwachs and L. Zuck (Eds.): TACAS 2005, LNCS 3440, pp. 301–316, 2005.

these logics and very few experimental results profile the proposed techniques. In this paper, we address both these issues.

In recent years, bounded model checking (BMC) [3] has emerged as a practically useful method, especially for detecting shallow bugs in complex systems. BMC exhaustively explores the system behaviour up to a bounded depth k in the state transition graph. Typically such exploration is reduced to solving satisfiability of a propositional formula. Inspired by the success of the BMC approach, we apply it to the problem of validity checking of IDL formulae. We consider the question '`Does there exist a model (timed state sequence) of length` k `that violates a given IDL formula?`' or '`Is the given IDL formula` k`-valid?`'. As in the BMC approach, we reduce the problem of checking k-bounded validity of an IDL formula to checking unsatisfiability of a *lin-sat* formula, which is a Boolean combination of propositional variables and mathematical constraints over real variables. We then use 'Integrated Canonizer and Solver (ICS)[1]' [7], a SAT-based solver to check the *lin-sat* formula for satisfiability.

As our primary contribution, we propose an efficient encoding of k-validity of IDL into unsatisfiability of *lin-sat*. Our encoding is is linear in the size of the IDL formula and cubic in k. Since the unsatisfiability of *lin-sat* is in Co-NP, this provides a Co-NP algorithm for deciding k-validity of IDL formula. We also give experimental evidence of the effectiveness of the proposed technique.

It must be noted that Fränzle [9] was the first to suggest bounded validity checking of a discrete-time duration calculus without timing constraints by polynomial-sized reduction to propositional SAT solving. In this paper, we extend Fränzle's techniques to deal with discrete and dense-time duration constructs in an efficient manner.

As our second contribution, we implement some alternative methods for checking the validity of IDL formulae. These are based on the digitization technique of Chakravorty and Pandya [6] combined with automata-theoretic analysis [14], as well as propositional SAT solving [9]. We provide experimental results on the relative performance of these techniques on several problems drawn from the Duration Calculus literature.

The remainder of this paper is organized as follows. In Section 2, we recall from [16] the basics of Interval Duration Logic. In Section 3, we present a polynomial-time reduction of the bounded validity checking problem for IDL formulae to the problem of checking unsatisfiability of *lin-sat* formulae, and prove the correctness of our reduction. In Section 4, we compare our approach with digitization based approaches and give experimental results. We end the paper with some conclusions and a discussion of related work in Section 5.

2 Interval Duration Logic: An Overview

Let $Pvar$ be the set of propositional variables. The set of states, Σ, is given by 2^{Pvar}, *i.e.*, the set of all subsets of $Pvar$. Let $\mathbb{R}$ denote the set of real numbers and $\mathbb{R}^0$ denote the set of non-negative real numbers.

[1] ICS is developed by SRI International.

Definition 1. *A* **timed state sequence** *over Pvar is a pair* $\theta = (\sigma, \tau)$*, where* $\sigma = s_0\ s_1\ \ldots\ s_{n-1}$ *is a finite non-empty sequence of states with* $s_i \in 2^{Pvar}$*, and* $\tau = t_0\ t_1\ \ldots\ t_{n-1}$ *is a finite non-decreasing sequence of time-stamps such that* $t_i \in \mathbb{R}^0$ *with* $t_0 = 0$*.*

A timed state sequence gives a sampled view of timed behaviour. It is assumed that the system evolves by discrete transitions. Let $dom(\theta) = \{0, \ldots, n-1\}$ be the set of positions within the sequence θ. Let the length, n, of θ be denoted by $\#\theta$. Let $\theta[i]$ denote the timed state at the i^{th} position of θ. 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 duration $[t_i, t_{i+1})$, which includes t_i but excludes t_{i+1}. The set of intervals in θ is given by $Intv(\theta) = \{[b, e] \in dom(\theta)^2 | b \leq e\}$, where each interval $[b, e]$ identifies a sub-sequence of θ between positions b and e. We use the notation $\theta, i \models P$ to denote that proposition P evaluates to true at state $\theta[i]$.

Syntax of IDL.[2] Let Π be a set of propositions over a finite set of propositional variables $Pvar$ and let c range over non-negative integer constants. The set of formulae of IDL is inductively defined as follows:

- $\eta \bowtie c$ and $\ell \bowtie c$ are formulae, where $\bowtie\ \in \{<, =, >, \geq, \leq\}$.
- If $P \in \Pi$ then $\lceil P \rceil^0$, $\lceil\lceil P \rceil$, $\sum P \bowtie c$, and $\int P \bowtie c$ are formulae.
- If D, D_1 and D_2 are formulae, then so are $D_1 \frown D_2$, $D_1 \wedge D_2$ and $\neg D$.

Semantics of IDL. The satisfaction of an IDL formula D for behaviour θ and interval $[b, e] \in Intv(\theta)$ is denoted as $\theta, [b, e] \models D$, and is defined as follows.

$\theta, [b, e] \models \lceil P \rceil^0$ **iff** $b = e$ and $\theta, b \models P$
$\theta, [b, e] \models \lceil\lceil P \rceil$ **iff** $b < e$ and $\theta, i \models P$ for all i such that $b \leq i < e$
$\theta, [b, e] \models \neg D$ **iff** $\theta, [b, e] \not\models D$
$\theta, [b, e] \models D_1 \wedge D_2$ **iff** $\theta, [b, e] \models D_1$ and $\theta, [b, e] \models D_2$
$\theta, [b, e] \models D_1 \frown D_2$ **iff** $\exists m : b \leq m \leq e : \theta, [b, m] \models D_1$ and $\theta, [m, e] \models D_2$

Entities η, ℓ, $\sum P$ and $\int P$ are called *measurements*. The entity η, called the *step length*, denotes the number of steps within a given interval, while the *time length* ℓ gives the amount of real-time spanned by a given interval. $\sum P$, called the *step count*, denotes the number of states for which proposition P is true within the interval $[b, e)$. *Duration* $\int P$ gives the amount of real-time for which proposition P holds in the given interval. η and $\sum P$ are called *discrete measurements*, while ℓ and $\int P$ are called *dense measurements*. The value of a measurement term t in a timed state sequence θ and an interval $[b, e]$, denoted as $eval(t, \theta, [b, e])$, is defined as follows:

- $eval(\eta, \theta, [b, e]) = e - b$
- $eval(\ell, \theta, [b, e]) = t_e - t_b$
- $eval(\sum P, \theta, [b, e]) = \sum_{i=b}^{e-1} \begin{cases} 1 & if \quad \theta, i \models P \\ 0 & otherwise \end{cases}$

[2] Here we consider the version of IDL without quantification $\exists p.D$.

– $eval(\int P, \theta, [b,e]) = \sum_{i=b}^{e-1} \begin{cases} t_{i+1} - t_i & if \quad \theta, i \models P \\ 0 & otherwise \end{cases}$

We say that $\theta, [b,e] \models t \bowtie c$ **iff** $eval(t, \theta, [b,e]) \bowtie c$.

Derived Operators. Let $\Diamond D \stackrel{\text{def}}{=} true \frown D \frown true$ and $\Box D \stackrel{\text{def}}{=} \neg\Diamond\neg D$. Then, $\Diamond D$ holds for an interval $[b,e]$ if for some sub-interval of $[b,e]$, the formula D holds. Similarly, $\Box D$ holds for $[b,e]$ if for all sub-intervals of $[b,e]$, the formula D is true.

Definition 2 (k-Bounded Validity of IDL Formulae). *Let $D \in IDL$, and let $k \in \mathbb{N}$ be a natural number. We define the following terminology.*

– **Validity in behaviour:** $\theta \models D$ **iff** $\theta, [0, \#\theta - 1] \models D$
– **Validity:** $\models D$ **iff** $\theta \models D$ *for all* θ
– **k-satisfiability:** $sat_k(D)$ **iff** $\theta \models D$ *for some* θ *such that* $\#\theta = k + 1$
– **k-validity:** $\models_k D$ **iff** $\theta \models D$ *for all* θ *such that* $\#\theta = k + 1$ □

Example 1 (Gas Burner). Consider a simplified version of the gas burner problem from [21]. Formula $des1 \stackrel{\text{def}}{=} \Box(\lceil\lceil Leak\rceil \Rightarrow \ell \leq maxleak)$ states that any occurrence of gas leakage, $Leak$, will be stopped (by turning off Gas in full design) within $maxleak$ seconds. Formula $des2 \stackrel{\text{def}}{=} \Box((\lceil\lceil Leak\rceil \frown \lceil\lceil \neg Leak\rceil \frown \lceil Leak\rceil^0) \Rightarrow \ell \geq minsep)$ states that between two occurrences of $Leak$ there will be at least $minsep$ seconds. In its full version, this is achieved by keeping Gas off for at least $minsep$ time. The safety requirement here is that "gas must never leak for more than $leakbound$ seconds in any period of at most $winlen$ seconds". This is captured by the formula $concl \stackrel{\text{def}}{=} \Box(\ell \leq winlen \Rightarrow \int Leak \leq leakbound)$. To establish the safety requirement, we must prove the validity of the following IDL formula for the given values of constants $maxleak$, $minsep$, $winlen$ and $leakbound$:
$G(maxleak, minsep, winlen, leakbound) \stackrel{\text{def}}{=} des1 \wedge des2 \Rightarrow concl$ □

2.1 Sub-logics and Decidability

Let DDC be the subset of IDL where dense-time measurement constructs of the form $\ell \bowtie c$ or $\int P \bowtie c$ are not used. Let DDCR be a further subset of DDC where even the discrete-time measurement constructs $\eta \bowtie c$ or $\Sigma P \bowtie c$ are not used. For DDC formulae, the time-stamps τ in behaviour $\theta = (\sigma, \tau)$ do not play any role. We can therefore define $\sigma \models D$. The following theorem establishes the decidability of DDC and DDCR.

Theorem 1. *[14] For every DDC formula D over propositional variables $Pvar$, we can effectively construct a finite state automaton $A(D)$ over the alphabet 2^{Pvar} such that for all state sequences $\sigma \in (2^{Pvar})^*$, $\sigma \models D$ **iff** $\sigma \in L(A(D))$. Hence, satisfiability (validity) checking of DDC and DDCR formulae are decidable, and can be reduced to checking the existence of an accepting (rejecting) path in $A(D)$.* □

A tool, called DCVALID, based on the above automata-theoretic decision procedure for DDC has been implemented earlier, and has been found to be effective on many significant examples [14]. Although the lower bound on the size of automaton $A(D)$ is non-elementary in the worst-case, such blowup is rarely observed in practice (see [14]).

If, however, we consider the full dense-time logic IDL, its rich expressive power comes at the cost of decidability.

Theorem 2. *[16] The satisfiability (and hence validity) checking of IDL formulae is undecidable .* □

In spite of this, for practical applicability, there has been interest in developing partial techniques for checking validity of IDL formulae. One such partial technique was the digitization approach of Chakravorty and Pandya [6]. In the next section, we present a new approach to deciding k-validity of IDL formulae.

3 From IDL to *lin-sat*

As mentioned earlier, a *lin-sat* formula is a Boolean combination of propositional variables and linear constraints over real variables. Each linear constraint is restricted to be in one of the forms: (a) $(x - y) \bowtie c$, (b) $\sum_{x_i \in V} x_i \bowtie c$ or (c) $(x - y) = z$, where x, x_i, y and z are real variables, V is a finite set of real variables, c is an integer constant and $\bowtie \in \{<, >, \geq, \leq, =\}$. For example, $(x_1 + x_3) \leq 3 \wedge (b_2 \vee (x_2 - x_3 = x_1))$ is a *lin-sat* formula.

Let the sets of propositional and real variables appearing in a *lin-sat* formula ϕ be denoted by $Pvar$ and $Rvar$ respectively. An *interpretation* $\mathcal{I}$ consists of (i) a mapping of variables in $Pvar$ to $\{\mathsf{True}, \mathsf{False}\}$ and (ii) a mapping of variables in $Rvar$ to $\mathbb{R}$. A *lin-sat* formula ϕ is *satisfiable* if there exists an *interpretation* for which ϕ evaluates to True. A *lin-sat* formula is *valid* if it is satisfiable for all interpretations. We denote this by $\models_{linsat} \phi$.

Theorem 3. *The satisfiability problem for lin-sat is NP-complete. Hence, validity (unsatisfiability) checking for lin-sat is co-NP-complete.*

Proof: Given a *lin-sat* formula ϕ, let $Constr(\phi)$ denote the set of syntactically distinct linear constraints in ϕ. For each constraint $\chi_i \in Constr(\phi)$, let v_{χ_i} be a propositional variable distinct from all variables in $Pvar$. The resulting set of variables, $\{v_{\chi_i} \mid \chi_i \in Constr(\phi)\}$, is denoted $Auxvar$. Let $\phi^a = \phi[v_{\chi_i}/\chi_i]$ be the propositional formula obtained by replacing each linear constraint χ_i in the formula ϕ by v_{χ_i}. Let μ be an assignment of truth values to variables in $Pvar \cup Auxvar$. We denote by $LinSys(\mu)$ the set $\{\chi_i \mid \mu(v_{\chi_i}) = \mathsf{True}\} \cup \{\neg\chi_j \mid \mu(v_{\chi_j}) = \mathsf{False}\}$. We call μ consistent if all constraints in $LinSys(\mu)$ are simultaneously satisfiable. It is straightforward to see that ϕ is satisfiable if and only if there is a consistent μ such that $\mu \models \phi^a$. Thus, we can non-deterministically guess an assignment μ, and then verify in polynomial-time that it is consistent and satisfies ϕ^a. Since the size of ϕ^a, henceforth denoted as $|\phi^a|$, is linear in $|\phi|$, checking if $\mu \models \phi^a$ requires time

linear in $|\phi|$. The check for consistency of μ reduces to determining the feasibility of the set, $LinSys(\mu)$, of linear constraints. Since the size of $LinSys(\mu)$ is linear in $|\phi|$, this check also requires time polynomial (grows as the fifth power) in $|\phi|$ [12]. Thus, satisfiability for *lin-sat* is in NP. To see that it is also NP-hard, we note that an arbitrary instance of 3-SAT is also an instance of satisfiability of *lin-sat*.

3.1 An Encoding Scheme

Given an IDL formula D and an integer bound $k \geq 0$, we now describe a technique to construct a *lin-sat* formula, $compile(D, k)$, such that D is k-valid *if and only if* $compile(D, k)$ is an unsatisfiable *lin-sat* formula.

We first define the sets of variables used in $compile(D, k)$, which are different from those used in D. (i) Let $Pvar$ be the set of propositional variables of D. Let $LSatPvar(k) = \{x_0, \ldots, x_k \mid x \in Pvar\}$, where x_i represents the value of x in state $\theta[i]$. Moreover, for a proposition P over $Pvar$, let P_i be obtained by replacing each x by x_i in P. Then, P_i represents the value of P in state $\theta[i]$. (ii) Let $LSatTvar(k) = \{t_0, \ldots, t_k\}$, where the t_i are fresh real variables representing time-stamps in the timed state sequence θ. (iii) For every proposition P that occurs in a measurement sub-formula, *i.e.*, $\int P \bowtie c$ and/or $\Sigma P \bowtie m$, we introduce k real variables, dur_P_i and/or c_Q_i for i in $0 \ldots (k-1)$. We call this set of variables as $LSatMvar(D, k)$.

In order to correctly capture the semantics of IDL in *lin-sat*, we need to introduce some constraints on the variables defined above. Specifically, time must not flow backward. Moreover, variables dur_P_i and c_P_i are intended to represent the values of $\int P$ and ΣQ in the interval $[i, i+1]$. Thus, we have the following invariants.

$$INVT(k) \stackrel{\text{def}}{=} (t_0 = 0) \wedge \bigwedge_{i=0}^{k-1} (t_i \leq t_{i+1})$$

$$INVD(k) \stackrel{\text{def}}{=} \bigwedge_{i=0}^{k-1} \left(\bigwedge_{c\_Q_i \in LSatMvar(D,k)} (Q_i \wedge (c\_Q_i = 1)) \vee (\neg Q_i \wedge (c\_Q_i = 0)) \right) \wedge$$

$$\bigwedge_{i=0}^{k-1} \left(\bigwedge_{dur\_P_i \in LSatMvar(D,k)} (P_i \wedge (dur\_P_i = t_{i+1} - t_i)) \vee (\neg P_i \wedge (dur\_P_i = 0)) \right)$$

$$FINV(k) \stackrel{\text{def}}{=} INVT(k) \wedge INVD(k)$$

Given an IDL formula D and an integer $k \geq 0$, we define the syntactic encoding of k-validity of D as unsatisfiability of the following *lin-sat* formula.

$$compile(D, k) \quad \stackrel{\text{def}}{=} \quad FINV(k) \wedge \neg \beta^{[0,k]}(D).$$

Here, $\beta^{[0,k]}(D)$ is computed recursively using the translation scheme shown in Table 1. In this table, column 2 gives an IDL sub-formula, D', and column 3 gives its encoding for subinterval $[b, e]$, where $0 \leq b \leq e \leq k$. For notational convenience, we denote this encoding as $\beta^{[b,e]}(D')$.

Table 1. Translation of IDL to *lin-sat*

No.	**IDL :** D'	***lin-sat***: $\beta^{[b,e]}(D')$
1	$\lceil P \rceil^0$	$P_b \ \wedge \ (b = e)$
2	$\lceil\lceil P \rceil\rceil$	$\bigwedge_{i=b}^{e-1} P_i \ \wedge \ (\ e > b)$
3	$\eta \bowtie m$	$(e - b) \bowtie m$
4	$\ell \bowtie m$	$(t_e - t_b) \bowtie m$
5	$\Sigma P \bowtie m$	$\sum_{i=b}^{e-1} \ c\_P_i \ \ \bowtie \ \ m$
6	$\int P \bowtie m$	$\sum_{i=b}^{e-1} \ dur\_P_i \ \ \bowtie \ \ m$
7	$\neg D_1$	$\neg(\beta^{[b,e]}(D_1))$
8	$D_1 \wedge D_2$	$\beta^{[b,e]}(D_1) \quad \wedge \quad \beta^{[b,e]}(D_2)$
9	$D_1 \frown D_2$	$\bigvee_{j=b}^{e} \left[\ (\beta^{[b,j]}(D_1)) \quad \wedge \quad (\beta^{[j,e]}(D_2)) \ \right]$

3.2 Proof Outline for Correctness of *lin-sat* Reduction

In this Section, we will use $\models_{lin}$ for satisfaction in *lin-sat* to distinguish it from $\models$ which denotes satisfaction in IDL. Given a timed state sequence θ of length $k+1$ and an IDL formula D, we restrict ourselves to *lin-sat* formulae over variables in $LSatPvar(k) \cup LSatTvar(k) \cup LSatMvar(D,k)$. We define $\mathcal{I}_\theta$ as an interpretation of *lin-sat* in which each propositional variable x_i in $LSatPvar(k)$ is assigned the value of x in state $\theta[i]$, and each t_i in $LSatTvar(k)$ is assigned the time-stamp τ_i of state $\theta[i]$. In addition, each variable c_P_i in $LSatMvar(D,k)$ is assigned 1 if $\theta, i \ \models P$, and is assigned 0 otherwise. Similarly, each variable dur_Q_i in $LSatMvar(D,k)$ is assigned $(\tau_{i+1} - \tau_i)$ if $\theta, i \ \models Q$, and is assigned 0 otherwise. Thus, by definition, $\mathcal{I}_\theta \ \models_{lin} \ FINV(k)$.

Similarly, given an interpretation $\mathcal{I}$ of *lin-sat* that satisfies $FINV(k)$, we define $\theta_\mathcal{I}$ as a timed state sequence in which (i) the value of propositional variable x in state $\theta_\mathcal{I}[i]$ is the same as that of x_i in $\mathcal{I}$, and (ii) the time-stamp of $\theta_\mathcal{I}[i]$ is equal to the value of t_i in $\mathcal{I}$, for all i in 0 through k.

Lemma 1. *There is a bijection between the set of timed state sequences, θ, of length $k+1$ and the set of interpretations, $\mathcal{I}$, of lin-sat that satisfy $FINV(k)$.*

The proof for the lemma follows from the fact that the definitions of $\mathcal{I}_\theta$ and $\theta_\mathcal{I}$ are injections. We will denote the bijective pair as $(\theta, \mathcal{I}_\theta)$ or $(\mathcal{I}\ , \theta_\mathcal{I})$, as convenient.

Theorem 4. *Let $D \in IDL$ and θ be a timed state sequence with $\#\theta = k+1$. For all $[b,e] \in Intv(\theta)$ we have $\theta, [b,e] \models D$* **iff** *$\mathcal{I}_\theta \ \models_{lin} \beta^{[b,e]}(D)$.*

Proof: The proof is by induction on the structure of D.

Base Cases: We prove only one case, that of $D = \ell \bowtie m$. The proof for the other cases, where D is $\lceil P \rceil^0$, $\lceil\lceil P \rceil\rceil$, $\eta \bowtie c$, $\sum P \bowtie c$ or $\int P \bowtie c$ are omitted for lack of space. The full proof can be found in [17].

– Let $D = \ell \bowtie m$. Then,

$\theta, [b, e] \models \ell \bowtie m$
iff $\tau_e - \tau_b \bowtie m$ . . . from IDL semantics
iff $\mathcal{I}_\theta \models_{lin} ((t_e - t_b) \bowtie m)$ . . . as $\mathcal{I}_\theta(t_b) = \tau_b$ and $\mathcal{I}_\theta(t_e) = \tau_e$.
iff $\mathcal{I}_\theta \models_{lin} \beta^{[b,e]}(D)$ . . . from Table 1

Induction Step: We prove only one case, that of $D = D_1 \frown D_2$. The proof for the other cases where D is $D_1 \wedge D_2$ or $\neg D_1$ are similar and are omitted for lack of space.

– Let $D = D_1 \frown D_2$. Then,

$\theta, [b, e] \models D_1 \frown D_2$
iff . . . from IDL semantics
for some $m : b \leq m \leq e$, $\theta, [b, m] \models D_1$ and $\theta, [m, e] \models D_2$
iff . . . by the induction hypothesis
for some $m : b \leq m \leq e$, $\mathcal{I}_\theta \models_{lin} (\beta^{[b,m]}(D_1))$ and $\mathcal{I}_\theta \models_{lin} (\beta^{[m,e]}(D_2))$
iff . . . from semantics of $\vee$
$\mathcal{I}_\theta \models_{lin} \bigvee_{m=b}^{e} \left[(\beta^{[b,m]}(D_1)) \wedge (\beta^{[m,e]}(D_2))\right]$
iff . . . from Table 1
$\mathcal{I}_\theta \models_{lin} \beta^{[b,e]}(D)$ □

Corollary 1. $\models_k D$ **iff** *compile*(D, k) *is unsatisfiable for all interpretations in lin-sat. Moreover,* $\mathcal{I} \models$ *compile*(D, k) **implies** $\theta_\mathcal{I} \not\models D$. □

3.3 Optimizing the Encoding

In the above encoding scheme, the same *lin-sat* sub-formula may be replicated at several places when generating $\beta^{[0,k]}(D)$ due to repeated evaluation of $\beta^{[i,j]}(D')$. This can lead to an exponential blowup in the size of the output formula (see [17, 9] for concrete instances). In order to address this problem, following Fränzle [9], we introduce auxiliary variables for denoting potentially common sub-formulae.

Let $SubForm(D)$ be the set of sub-formulae of D. Then, $LSatAux(D, k) = \{\gamma_\psi^{[b,e]} \mid 0 \leq b \leq e \leq k, \psi \in SubForm(D)\}$ gives the set of auxiliary variables used for the optimized encoding. We now reuse the notation of Table 1, and modify column 3 of rows numbered 7, 8 and 9 in this table by replacing all occurrences of $\beta^{[b,e]}(D_i)$ with $\gamma_{D_i}^{[b,e]}$. Let the new encoding scheme, represented by the modified table, be called $\delta^{[b,e]}(D)$. Given an IDL formula D and an integer $k > 0$, the optimized *lin-sat* encoding can now be obtained as

$$compile\_opt(D, k) \stackrel{\text{def}}{=} FINV(k) \wedge \neg \gamma_D^{[0,k]} \wedge \bigwedge_{\gamma_\psi^{[b,e]} \in LSatAux(D,k)} (\gamma_\psi^{[b,e]} \Leftrightarrow \delta^{[b,e]}(\psi)).$$

It is straightforward to see that $compile\_opt(D,k)$ is equisatisfiable to the formula $compile(D,k)$, and therefore the correctness proof of Section 3.2 applies here as well. The number of auxiliary variables in $LSatAux(D,k)$ is $O(k^2.|D|)$, and the worst-case size of $\delta^{[b,e]}(\psi)$ is $O(k)$ (from rows 2, 5, 6 and 9 of Table 1). Also, the size $|FINV(k)|$ is $|INVT(k)|+|INVD(k)|+1$. From the definitions of these invariants, $|INVT(k)|$ is $O(k)$ and $|INVD(k)|$ is $O(k.|D|)$. Thus, the size of $compile\_opt(D,k)$ is $O(k^3.|D|)$.

As a further optimization, we note that auxiliary variables need not be introduced for every sub-formula and sub-interval combination. Therefore, we have implemented our encoding tool, *idl2ics*, as a two-pass translator. In the first pass, we identify all sub-formulae and sub-interval combinations that repeat in the encoding scheme, and introduce auxiliary variables only for these combinations. This effectively reduces the size of $LSatAux(D,k)$. In the second pass, the translator uses these auxiliary variables to generate $compile\_opt(D,k)$.

3.4 A Comparison of IDL Validity Checking Approaches

In this section, we give a comparative overview of several approaches proposed in the literature for checking (bounded) validity of IDL formulae.

Approach A: This is the bounded validity checking approach proposed in Corollary 1 of this paper. Checking k-validity of an IDL formula D is polynomially reduced to checking unsatisfiability of the *lin-sat* formula $compile\_opt(D,k)$. The size of $compile\_opt(D,k)$ is at most $O(k^3.|D|)$ where $|D|$ includes the size of binary encoding of integer constants occurring in D. Formula $compile\_opt(D,k)$ can be checked for unsatisfiability using a Co-NP algorithm based on Theorem 3.

Theorem 5. *Using approach A, k-validity checking of IDL formula D is decidable with Co-NP complexity. The complement problem, that of finding a satisfying assignment of $compile\_opt(D,k)$, is solvable by an NP algorithm with input size $O(k^3.|D|)$, i.e. polynomial in both k and $|D|$.* □

Approach B: Chakravorty and Pandya [6] have proposed a *digitization technique* for reducing validity checking of IDL formulae to validity checking of formulae in the discrete-timed logic DDC (see Section 2.1). They have defined a translation $dig : IDL \rightarrow DDC$ such that $\models_{DDC} dig(D)$ implies $\models_{IDL} D$. They have also proposed simple structural tests to identify a subclass $SYNCID \subset IDL$ for which $\models_{DDC} dig(D)$ if and only if $\models_{IDL} D$. The size of $dig(D)$ is $O(|D|)$, and validity of $dig(D)$ can be checked using the automata-theoretic decision procedure implemented in the tool DCVALID [14]. The worst case complexity of validity checking of DDC formulae is non-elementary in the size of the formula.

Theorem 6. *Using approach B, the validity of $D \in SYNCID(\subset IDL)$ is decidable by an algorithm with non-elementary worst-case complexity.* □

Approach C: Recall from Section 2.1 that the subset $DDCR$ of DDC consists of formulae without any quantitative measurements. It has been shown by earlier

researchers [14, 8, 19] that every formula $D \in DDC$ can be effectively transformed to an equivalent formula $untime(D) \in DDCR$. The worst-case size of $untime(D)$ is $O(2^{|D|})$, where $|D|$ includes the size of binary encoding of constants occurring in D. In approach C, a formula $D \in SYNCID(\subset IDL)$ is first digitized to a validity preserving formula $dig(D) \in DDC$. This is then reduced to an equivalent DDCR formula, $untime(dig(D))$, which can be checked for bounded validity. Note that for $D' \in DDCR$, the translation $compile\_opt(D', m)$ gives a purely propositional formula whose unsatisfiability can be established by *propositional SAT-solving*. The size of the propositional formula $compile\_opt(untime(dig(D)), m)$ is $O(m^3.|untime(dig(D))|)$, which is $O(m^3.2^{|D|})$ in the worst-case. Fränzle [9] first suggested checking m-validity of DDC formulae using propositional SAT solving. Hence, approach C is an extension of his work.

In the context of bounded validity checking, the effect of digitization on the length of counter-models is an important factor to consider. Let D be digitizable, i.e. $D \in SYNCID$. Chakravorty and Pandya [6] have shown that every timed state sequence θ of logic IDL can be represented by a state sequence $\hat{\theta}$ of DDC such that $\theta \models_{IDL} D$ **iff** $\hat{\theta} \models_{DDC} dig(D)$. The encoding of time in $\hat{\theta}$ is achieved by having exactly one unit of time elapse between successive states in the sequence. In contrast, the elapse of an arbitrary length of time between successive states can be represented in the timed state sequence θ by using appropriate time-stamps. Thus, a discretized (counter-)model $\hat{\theta}$ is typically much longer than the corresponding time-stamped (counter-)model θ. Let k be the length of the shortest (counter-)model of D and let k' be the length of shortest (counter-)model of $dig(D)$. While it is difficult to estimate k' directly from k and D, it is easy to see that $k \leq k'$. This follows from the fact that the bijective mapping $\theta \rightarrow \hat{\theta}$ does not reduce the length of the model [6]. We will present experimental results comparing k and k' on several benchmark problems later in the paper.

Theorem 7. *Using approach C, k-validity of $D \in SYNCID\,(\subset IDL)$ is decidable with Co-NEXP complexity. The complement problem, that of finding a satisfying assignment of $compile\_opt(untime(dig(D)), k')$ is solvable by an NP algorithm with input size $O(k'^3.2^{|D|})$. i.e., polynomial in k' (with $k' \geq k$) and* ***exponential*** *in $|D|$.* □

4 Implementation and Experimental Results

The three approaches outlined in the previous section have widely differing theoretical complexities. Moreover, each of them gives a partial solution to the problem of checking whether an IDL formula D is valid. Hence, an experimental evaluation of their effectiveness and efficiency is needed.

Implementation: For realizing approach A, we have implemented a translator, *idl2ics*, from IDL to *lin-sat*, giving the formula $compile\_opt(D, k)$. The resulting *lin-sat* formula is checked for unsatisfiability using the ICS solver [7]. Using Corollary 1, if a satisfying assignment is found by ICS, a counter-example for the original IDL formula is obtained in an encoded form. If the *lin-sat* formula is found to be

unsatisfiable, the original IDL formula is declared k-valid. We have also conducted preliminary experiments with other solvers like UCLID and CVClite for checking unsatisfiability of *lin-sat* formulae. However, ICS significantly outperformed both CVClite and UCLID for our benchmarks [17]. Hence our detailed experiments were conducted with the ICS solver.

For implementing digitization-based approaches B and C, we have developed a translator, $idl2ddc$, that takes a formula $D \in IDL$ and returns $dig(D)$ along with an indication of whether $D \in SYNCID$. In approach B, the formula $dig(D)$ is checked for validity using the DCVALID tool. In approach C, we further translate $dig(D)$ to $untime(dig(D))$ using a translator, $ddc2ddcr$, that transforms a formula $D \in SYNCID$ to a validity-preserving formula $untime(D) \in DDCR$. The formula $untime(dig(D))$ is then checked for bounded validity by translating it using $idl2ics$ with a suitable bound k' for the model length. The resulting propositional formula is then checked for unsatisfiability using the ICS solver.

We draw our set of benchmarks from the Duration Calculus literature. We refer the reader to [11, 17] for the detailed IDL specification of each problem. Each problem was formulated such that the IDL formula is in $SYNCID$. Owing to space limitations, we simply list the formula name for each problem here along with a list of its parameters (time constants) below. In each case, the aim is to check the validity of the IDL specification for some given values of time constants.

1. G($maxleak, minsep, winlen, leakbound$) denotes a **gas burner** specification [21], given earlier as Example 1.
2. M($\delta, \omega, \epsilon, \zeta, \kappa$) denotes a **minepump controller** specification [16, 13].
3. L(t_s, t_0, t_{max}, t_m) denotes a **lift controller** specification [4, 17].
4. J24 and J44 denote **job-shop scheduling** problems on 4 machines with 2 and 4 jobs respectively [17].

Experimental Results: Table 2 gives our experimental results comparing the performance of the three approaches. Here, "CE len" denotes the length of the smallest counter-example found by the various approaches. In case the formula is valid (as detected by approach B), we use the entry V for "CE len". For such formulae, we do not apply approaches A and C, since these bounded validity checking techniques are useful only for invalid formulae. The corresponding rows in Table 2 are therefore marked "Valid - BVC Not Applied". The various columns under each approach show the translation times and the time taken to check validity (unsatisfiability) of the resulting formulae using ICS or DCVALID. In Approach B, a "↓" entry denotes an abort due to the BDD-nodes for the automata representation exceeding a fixed threshold. An entry `>26h` denotes that the corresponding computation was aborted after 26 hours since it did not terminate in that time. A "-" entry denotes non-applicability due to the failure of the preceding stage. Counter-example generation is not applicable to valid formulae and this is denoted by an 'NA" entry. All our experiments were performed on a 1 GHz i686 PC with 1GB RAM running RedHat Linux 7.3.

Note on DCVALID: In approach B above, the validity of $dig(D)$ is checked using the tool DCVALID [14]. In Table 2, column 3 under approach B gives the time taken

Table 2. Comparative Performance of IDL Validity Checking Approaches

Example	Approach A			Approach C				Approach B					
	CE len	idl2*lin-sat* +ICS time (secs)		CE len	idl2ddcr + ddcr2sat + ICS time(secs)			CE len	idl2ddc + Dcvalid time(secs)		OPT-CTLDC time(secs)		
											T	R	C
M(2,12,5,20,2)	7	0.01	2.28	-	0.01	>26h	-	30	0.01	39.62	1.76	2.99	5m35
M(20,120,50,100,20)	7	0.02	2.30	-	0.06	>24h	-	-	0.01	↓	↓	-	-
M(3,25,5,27,5)	Valid - BVC not applicable							V	0.01	2m	2.08	1.52	NA
M(4,30,5,35,7)	Valid - BVC not applicable							V	0.01	↓	2.48	1.77	NA
G(1,2,11,3)	7	0.1	0.52	8	0.0	33.87	25.58	8	0.0	18.24	0.37	0.08	0.15
G(1,2,15,4)	9	0.0	4.41	10	0.1	1m17	16m34	10	0.0	↓	0.43	0.11	0.20
G(10,5,50,30)	7	0.01	1.00	-	0.02	>24h	-	35	0.01	↓	3.82	2.03	2.81
G(15,10,80,35)	5	0.0	0.22	-	0.0	>24h	-	39	0.01	↓	35.07	20.75	28.49
G(20,10,100,50)	5	0.0	0.22	-	0.0	>24h	-	-	0.01	↓	↓	-	-
G(20,10,70,45)	5	0.0	0.21	-	0.0	>24h	-	-	0.01	↓	↓	-	-
G(1,4,12,4)	Valid - BVC not applicable							V	0.0	↓	0.43	0.11	NA
L(10,3,5,10)	2	0.01	0.09	13	0.02	0.19	231m	-	0.01	↓	9.9	>1h	-
L(100,30,50,70)	3	0.02	0.12	-	0.08	>24h	-	-	0.0	↓	↓	-	-
J24	7*	0.01	0.30	-	0.02	>19h	-	33	0.0	10.01	1.29	8.44	3m18
J44	10*	0.05	259m	-	0.01	>24h	-	-	0.0	↓	2.95	1m4	>9h

Table 3. Growth of Computation time (in seconds) with k in k-Validity Checking

k	Mine1	Mine2	Gas1	Gas2	Gas3	Gas4	Gas5	Lift1	Lift2	Job24	Job44
1	0.10	0.08	0.09	0.09	0.09	0.09	0.08	0.08	0.09	0.09	0.09
2	0.10	0.10	0.08	4.73	0.08	0.09	0.09	**0.09**	0.10	0.09	0.10
3	0.17	0.16	0.07	4.73	0.12	0.12	0.13	0.12	**0.12**	0.09	0.12
4	0.32	0.29	0.08	4.74	0.18	0.16	0.17	0.15	0.17	0.12	0.16
5	0.81	0.58	0.08	4.79	0.33	**0.22**	**0.22**	-	0.23	0.20	0.26
6	1.31	1.07	0.67	4.67	0.71	0.25	0.25	-	-	0.48	0.69
7	**2.28**	**2.30**	**0.52**	1.68	**1.00**	0.30	0.29	-	-	**0.3**	2.96
8	6.47	4.96	0.65	4.43	0.77	-	-	-	-	0.28	15.8
9	10.02	9.98	0.75	**4.41**	0.84	-	-	-	-	0.63	8m23.65
10	-	-	-	5.38	-	-	-	-	-	-	**258m66.69**
11	-	-	-	6.47	-	-	-	-	-	-	Aborts
12	-	-	-	-	-	-	-	-	-	-	Aborts
Sum upto CE	5.09	4.58	1.59	29.86	2.51	0.68	0.69	0.17	0.31	1.37	16070.52

by the currently released version (1.4) of DCVALID. However, this tool suffers from several inefficiencies and a more optimized version of the automata-theoretic analysis has been implemented by us. This is denoted by "OPT-CTLDC" and

columns 4-6 under approach B of Table 2 give the time taken using OPT-CTLDC. The optimized analysis works in the following steps: (i) it applies a few validity preserving transformations to the QDDC formula, (ii) it translates the formula into a synchronous product of several sub-automata (taking time **T**), which are output as SMV modules [15], (iii) it performs reachability analysis of the SMV model (taking time **R**) to detect a counter-example of the original formula, and (iv) it performs a forward search (taking time **C**) to explicitly generate the shortest counter-example, if one exists. Step (ii) is performed using the tool CTLDC [15, 14] and steps (iii) and (iv) are performed using the tool NuSMV. Details of this optimized approach will be addressed in a future paper. However, it can be seen from Table 2 that the OPT-CTLDC based approach is significantly more efficient vis-a-vis the DCVALID based approach.

Table 3 profiles the time taken by ICS for checking satisfiability of formula $compile\_opt(k, D)$ for different values of k, as used in approach A. The entries in bold correspond to the shortest counter-example. It can be seen that the computation time grows smoothly with the value of k with no significant jump between valid and invalid instances.

5 Conclusions and Discussion

The dense-time logic IDL is undecidable in general (see Theorem 2). Hence, all algorithmic approaches to its validity checking are doomed to be incomplete. In this paper, we have presented a technique for bounded validity checking of IDL by reducing it to checking unsatisfiability of *lin-sat*. We have compared this technique to methods based on digitization of IDL, both theoretically and experimentally. Our experimental comparison used a variety of well-known examples. In the process, we have created tools [11] which allow such examples to be verified. We now discuss the relative merits of the three approaches (A, B and C) based on the results of Sections 3.4 and 4, and draw some conclusions.

Digitization (used in approaches B and C) reduces the validity of dense-time IDL to the validity of discrete-time DDC. While it is sound for all formulae (i.e. $\models_{DDC} dig(D) \Rightarrow \models_{IDL} D$), it is complete only for the sub-class $SYNCID \subset IDL$. We believe that this is not a crippling restriction in practice (see [6]).

In approach B, the digitized formula is checked for validity using the automata-theoretic decision procedure for DDC (see Theorem 6) implemented in the tool DCVALID as well as in its optimized version, OPT-CTLDC. While the worst-case complexity of this approach is non-elementary, this is rarely observed in practice. As shown in Table 2, this approach succeeds in proving validity/invalidity of several examples when the constants in the formulae are relatively small. Moreover, this approach can establish unbounded validity of a formula, while approaches A and C can only check bounded validity. Unfortunately, for many problem instances with large constants, approach B fails to succeed within reasonable time and space constraints.

Approach A checks k-validity of an IDL formula D by checking unsatisfiability of a *lin-sat* formula (see Theorem 5). While it cannot establish the validity of

Table 4. Gas Burner for different constants using Approach A

Constants	CE Len	Translation Time(secs)	ICS Time(secs)
G(5, 7, 69, 28)	11	0.01	18.43
G(10, 15, 137, 53)	11	0.01	18.24
G(210, 534, 4000, 1225)	11	0.01	18.39
G(7400, 9535, 93010, 44341)	11	0.01	18.66

an IDL formula D, like bounded model checking, it is useful for finding shallow counter-examples of invalid IDL formulae. As Table 2 shows, when used in this fashion, approach A is much more effective compared to the other approaches. Counter-examples (timed state sequences) of lengths up to 11 could be found relatively efficiently for almost all our benchmarks. When large constants are present in the formula, approach A clearly outperforms the other approaches. This is because the time taken for *lin-sat* solving is relatively insensitive to the scaling of constants. This is clearly borne out in Table 4. In contrast, digitization-based approaches, like B and C, that model the passage of time in steps of one unit, are very sensitive to scaling of constants.

Approach C combines digitization and SAT solving for bounded validity checking (see Theorem 7). Theoretically, it suffers from an exponential increase in the worst-case complexity over approach A. However, it uses propositional SAT solving instead of the more complicated *lin-sat* solving of approach A. Despite this, our experiments show that in most cases, approach C is not very effective and is outperformed by approaches A and B (OPT-CTLDC). One reason for this is the exponential increase in the size of the propositional SAT formula compared to the *lin-sat* formula of approach A. Another significant factor is the need for using a larger bound, k', for finding the shortest counter-example in approach C, as compared to the bound k used in approach A. The experimental results on the shortest counter-example lengths in Table 2 clearly point to this factor.

While we have presented a technique for bounded validity checking of IDL formulae, this can be easily extended to perform bounded model-checking of timed-automata [1] against an IDL specification D. Following the approach of Audemard *et al* [1], this can be achieved by checking the unsatisfiability of $compile\_opt(D, k)$ conjuncted with a *lin-sat* formula representing the k-step behaviour of the timed automaton. We propose to extend our tool with this capability in the future.

Comparison with Related Work: Bounded model checking of LTL formulae using SAT solving was proposed by Bierre *et al* [3] as an efficient method for finding shallow counter examples. Audemard *et al* [1] extended this to timed systems using MATHSAT solving. Fränzle [9] first proposed bounded validity checking of Discrete Duration Calculus without timing constructs (i.e., the same as DDCR) by a polynomial-sized reduction to propositional SAT solving. When used with the timing constructs ΣP and η, this reduction has an exponential blowup, assuming binary encoding of constants in the formula (see Theorem 7). Fränzle demonstrated that for simple instances of the discrete-time version of the gas burner problem, his

technique was superior to the automata-theoretic procedure DCVALID. In this paper, we have extended Fränzle's technique to the dense-time logic IDL including the *duration* and *count* constructs. We have given an encoding of k-validity of IDL into unsatisfiability of *lin-sat*, where the size of the encoding is polynomial in the size of the IDL formula D, with constants encoded in binary. We believe that our generalization of Fränzle's work is practically significant and advantageous, as demonstrated by our experimental evaluation. More recently, Fränzle and Herde have investigated efficient SAT-solving techniques for 0-1 linear constraints which arise in the translation of the discrete-time *count* construct [10].

Fränzle was influenced by the prior work of Ayari and Basin [2], who gave a polynomial-time encoding of the logic ML2STR (monadic logic of finite words) into Quantified Boolean Formulae for bounded validity checking. Ayari and Basin demonstrated that on many problems, the automata-theoretic decision procedure for ML2STR (using the MONA tool) performed better than the QBF SAT solving technique. But on some complex problems, QBF SAT solving was able to find counter-examples faster. Approach B in our experiments uses a similar automata-theoretic technique, but it handles the dense-time logic IDL using digitization. Moreover, the tool OPT-CTLDC considerably improves the automata based analysis.

Acknowledgments. The authors thank Martin Fränzle and A. Cimatti for their helpful comments, and Dina Thomas and S. N. Krishna for their help in conducting experiments.

References

1. G. Audemard, A. Cimatti, A. Kornilowicz, and R. Sebastiani. Bounded Model Checking for Timed Systems. In *FORTE*, volume 2529 of *LNCS*. Springer, 2002.
2. A. Ayari and D. Basin. Bounded Model Construction for Monadic Second-Order Logics. In *CAV*, volume 1855 of *LNCS*. Springer, 2000.
3. A. Bierre, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic Model Checking without BDDs. In *TACAS*, volume 1579 of *LNCS*. Springer, 1999.
4. Dines Bjørner. Trusted computing systems: The procos experience. In *ICSE*, pages 15–34, 1992.
5. A. Bouajjani, Y. Lakhnech, and R. Robbana. From Duration Calculus to Linear Hybrid Automata. In *CAV*, volume 939 of *LNCS*. Springer, 1995.
6. G. Chakravorty and P.K. Pandya. Digitizing Interval Duration Logic. In *CAV*, volume 2725 of *LNCS*. Springer, 2003.
7. J. Filliâtre, S. Owre, H. Rueß, and N. Shankar. ICS: Integrated Canonizer and Solver. In *CAV*, volume 2102 of *LNCS*. Springer, 2001.
8. M. Fränzle. Model-Checking Dense-Time Duration Calculus. In *M.R. Hansen (ed.), Duration Calculus: A Logical Approach to Real-Time Systems Workshop proceedings of ESSLLI X*, 1998.
9. M. Fränzle. Take it NP-easy: Bounded Model Construction for Duration Calculus. In *FTRTFT*, volume 2469 of *Lecture Notes in Computer Science*. Springer, 2002.
10. M. Fränzle and C. Herde. Efficient SAT engines for concise logics: Accelerating proof search for zero-one linear constraint systems. In M. Vardi and A. Voronkov, editors, *LPAR*, volume 2850 of *LNAI*, pages 302–316. Springer, 2003.

11. IDLVALID: Model Checking Dense-time Duration Logics. WWW page, 2004. http://www.tcs.tifr.res.in/~pandya/idlvalid.html.
12. N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.
13. Z. Liu. Specification and Verification in the Duration Calculus. In M. Joseph, editor, *Real-time Systems: Specification, Verification and Analysis*, pages 182–228. Prentice Hall, 1996.
14. P.K. Pandya. Specifying and Deciding Quantified Discrete-Time Duration Dalculus Formulae using DCVALID. In Paul Pettersson and Wang Yi, editors, *RTTools*, Uppsala University Technical Report Series, 2000.
15. P.K. Pandya. Model checking CTL[DC]. In *TACAS*, volume 2031 of *LNCS*. Springer, 2001.
16. P.K. Pandya. Interval Duration Logic: Expressiveness and Decidability. In E. Asarin, O. Maler, and S. Yovine, editors, *TPTS'02*, volume 65 of *ENTCS*. Elsevier Science Publishers, 2002.
17. B. Sharma. *SAT Based Validity Checking of Interval Duration Logic Formuale*. M.Tech Dissertation, Dept. of Computer Science and Engineering, IIT Bombay, June 2004.
18. J.U. Skakkebæk and P. Sestoft. Checking Validity of Duration Calculus Formulas. ESPRIT project PROCOS II. Technical report, Department of Computer Science, Technical University of Denmark, 1996.
19. Chaochen Zhou and M.R. Hansen. *Duration Calculus*. Springer, 2003.
20. Chaochen Zhou, C.A.R. Hoare, and A.P. Ravn. A Calculus of Durations. *Information Processing Letters*, 40(5):269–276, 1991.
21. Chaochen Zhou, A. Ravn, and M.R. Hansen. An Extended Duration Calculus for Hybrid Real-Time Systems. In *Hybrid Systems*, pages 36–59. Springer, 1993.