Formal Aspects of Computing (1997) 9: 283–330

Formal Aspects of Computing

Duration Calculus: Logical Foundations

Michael R. Hansen[1] and Zhou Chaochen[2]

[1]Department of Information Technology, Technical University of Denmark
[2]International Institute for Software Technology, United Nations University, Macau
on leave of absence from Software Institute, Academia Sinica, Beijing

Keywords: Duration Calculus; Interval Logic; Real-time systems; Formal methods

Abstract. The Duration Calculus (abbreviated DC) represents a logical approach for formal design of real-time systems, where real numbers are used to model time and Boolean valued functions over time are used to model states and events of real-time systems. Since its introduction, DC has been applied to many case studies and it has been extended in several directions. The aim of this paper is to provide a thorough presentation of the logic.

1. Motivation

A real-time system is a computing system with real-time requirements. Let us consider the following example of a real-time system.

1.1. An Example

Gas Burner: This example was first investigated by [SRR90]. A gas burner is either heating when the flame is burning or idling when the flame is not burning, and it alternates indefinitely between heating and idling. Usually, no gas is flowing while it is idling. However, when changing from idling to heating, gas must be flowing for a short time before it can be ignited, and when a flame failure appears, gas must be flowing before the failure is detected and the gas valve is closed. Hence, there may exist a time where gas is flowing and a flame is not burning, i.e. gas is *leaking*. A design of a safe gas burner must ensure that the time intervals where gas is leaking do not get too long.

Correspondence and offprint requests to: Michael R. Hansen, Department of Information Technology, Building 344, Technical University of Denmark, DK-2800 Lyngby, Denmark. E-mail: mrh@it.dtu.dk

Let us assume that the ventilation required for normal combustion would prevent dangerous accumulation of gas, provided that the proportion of leak time is not more than one twentieth of the elapsed time for any time interval being at least one minute long — otherwise the requirement would be violated immediately at the start of a leak. This is a real-time requirement.

Certain design decisions must be made as to how the real-time requirement is to be met. For example, for any period where the requirement is guaranteed, any leak should be detectable and stoppable within one second; and to prevent frequent leaks it is acceptable that after any leak the gas burner rejects switching on gas for thirty seconds. The conjunction of these two decisions implies the original requirement, a fact which should be proved before implementation proceeds. □

The gas burner is a real-time system and an example of a software embedded system, also called a hybrid system.

Duration Calculus is a logical approach for formal design of real-time systems. Real numbers are used to model time, and functions from time to Boolean values are used to model the behaviour of real-time systems. Based on interval logic [HMM83, Mos83], DC provides a formal notation to specify properties of real-time systems and a calculus to prove those properties formally, such as the correctness of the design decisions for the gas burner.

1.2. State Models

The notion *state* is used to model behaviour of real-time systems. A *Boolean state model* of a real-time system is a set of Boolean valued functions over time:

$$\mathbb{T}\text{ime} \rightarrow \{0,1\}$$

where $\mathbb{T}$ime is the set of the real numbers. Each Boolean valued function, also called a *Boolean state* (or simply a *state*) of the system, is a *characteristic* function of a *specific aspect* of the system behaviour, and the whole set of Boolean valued functions characterizes all the concerned aspects of the behaviour.

Gas Burner: To verify the design decisions against the requirement, one may start with a single Boolean state to model the critical aspect of the system:

$$\text{Leak} \in \mathbb{T}\text{ime} \rightarrow \{0,1\}$$

where $\text{Leak}(t) = 1$ means that gas is leaking at time t, and $\text{Leak}(t) = 0$ means that gas is not leaking at t. However, at a later stage of the design one may have to specify the phases of burning and idling of the gas burner, and introduce more primitive Boolean states of the system such as Gas and Flame to characterize flowing and burning of gas. Then Leak can be pointwise defined as the Boolean expression of Gas and Flame:

$$\text{Leak}(t) \mathrel{\widehat{=}} \text{Gas}(t) \wedge \neg\text{Flame}(t), \text{ for any } t \in \mathbb{T}\text{ime}$$

□

Boolean operators (e.g. $\neg$ and $\wedge$) for states are included in DC, so that a composite state of a real-time system can be refined to primitive states of the system.

We are interested in *non-Zeno states*, i.e. states changing at most a finite number of times in any finite interval. Therefore, we assume that Boolean states are *finitely varied*: P has at most a finite number of discontinuity points in $[b, e]$

for any Boolean state P and interval $[b, e]$. It is easy to see that the set of finitely varied Boolean states are closed under the Boolean operations, and that any Boolean state in the set is integrable in any interval $[b, e]$.

We are also interested in *stable* aspects of systems, e.g. when we observe a gas leak at time t, then gas leaks during a left and/or a right neighborhood of t. Therefore, a suitable restriction could be to consider Boolean states which are left or right continuous at any time t.

This set of left or right continuous functions is, however, not closed under the Boolean operations, e.g. for

$$\text{Gas}(t) = \begin{cases} 0 & \text{for } t < 2 \\ 1 & \text{for } t \geqslant 2 \end{cases}$$

$$\text{Flame}(t) = \begin{cases} 0 & \text{for } t \leqslant 2 \\ 1 & \text{for } t > 2 \end{cases}$$

we have that

$$\text{Leak}(t) = \text{Gas}(t) \wedge \neg\text{Flame}(t) = \begin{cases} 1 & \text{for } t = 2 \\ 0 & \text{for } t \neq 2 \end{cases}$$

Thus, Leak(t) is neither left nor right continuous at $t = 2$ despite the fact that both Gas and Flame are left or right continuous functions. Therefore, we only assume here the finite variability of Boolean states; but we will return to the discussion of left or right continuous states in Section 3.2

1.3. State Durations

The notion *state duration* is an essential measurements of the behaviour of real-time systems. The duration of a Boolean state over a time interval is the accumulated time in which the state is present in the interval. Let $P \in \mathbb{T}\text{ime} \to \{0, 1\}$ be a Boolean state and $[b, e]$ an interval, i.e. $b, e \in \mathbb{T}\text{ime}$ and $e \geqslant b$. Mathematically, the duration of state P over $[b, e]$ equals the integral

$$\int_b^e P(t)dt.$$

Let us use the gas burner example to illustrate the importance of state durations in specifications of real-time behaviour.

Gas Burner: The real-time requirement of the gas burner is that the proportion of leak time in an interval should not be more than one twentieth of the interval length, if the interval is at least one minute long. This requirement can be expressed in terms of the duration of Leak:

$$(e - b) \geqslant 60 \text{ sec.} \Rightarrow 20\int_b^e \text{Leak}(t)dt \leqslant (e - b) \text{ for any interval } [b, e]$$

□

Since gas leaks due to *random* flame failures, the duration of a leak must be used to extract the accumulated leak time of gas. Therefore, a mathematical formulation of this requirement can hardly leave out state durations. Hence, state duration is adopted in DC as an essential measurement of the behaviour of real-time systems.

1.4. State Distances

The *distance* between states (or events) is another important measurement of the behaviour of real-time systems and was extensively studied before the development of DC, e.g. in Timed Automata [AlD92], Real-Time Logic [JaM86], Metric Temporal Logic [Koy90], Explicit Clock Temporal Logic [HLP90], and Interval Temporal Logic [HMM83, Mos85b]. However, state durations are more expressive than state distances in the sense that the latter can be expressed in terms of the former, but not vice versa.

One can first express an occurrence of a state using state durations. Let us assume that a presence of state P lasts for a period of $[c,d]$ $(d > c)$. It can be expressed as the duration of P in $[c,d]$ is equal to the length of $[c,d]$:

$$\int_c^d P(t) = (d-c) > 0, \qquad \text{(abbreviated } P[c,d])$$

if we do not care about instant absence of P. In real analysis it is read as: "P appears almost everywhere in $[c,d]$". Thus, real-time constraints on occurrences of states can be expressed in terms of state durations.

Gas Burner: Consider the first design decision concerning the gas burner. Let $[b,e]$ be an arbitrary interval where we want to guarantee the requirements of the gas burner. The first design decision is that any leak in $[b,e]$ should not last for a period longer than one second. It can be expressed as:

$$\forall c,d : b \leqslant c \leqslant d \leqslant e.(\text{Leak}[c,d] \;\Rightarrow\; (d-c) \leqslant 1 \; sec.)$$

□

Similarly, real-time constraints on state distances can be expressed in terms of state durations.

Gas Burner: The second design decision concerning the gas burner is that the distance between any two consecutive occurrences of leaks during the interval $[b,e]$ must be at least thirty seconds long:

$$\forall c,d,f,g : b \leqslant c \leqslant d \leqslant f \leqslant g \leqslant e.$$
$$(\text{Leak}[c,d] \;\wedge \text{NonLeak}[d,f] \wedge \text{Leak}[f,g]) \;\Rightarrow\; (f-d) \geqslant 30 \; sec.$$

where NonLeak is a state defined from Leak using negation ($\neg$):

$$\text{NonLeak}(t) \;\widehat{=}\; \neg\text{Leak}(t), \text{ for any } t \in \mathbb{T}\text{ime}$$

□

Since state durations are more expressive than state distances, DC is more expressive than the existing formal approaches to real-time systems. By axiomatizing integrals of Boolean valued functions, DC also exhibits a possibility to introduce notions of real analysis into formal techniques for designing software embedded real-time systems. Nowadays one can find a notion of *integral* and/or *differential* in Automata [ACH93, NOS93], StateCharts [MaP93], Temporal Logic of Actions [Lam93], and Communicating Sequential Processes [He94].

State durations as integrals of Boolean valued functions are functions from time intervals to real numbers. It is therefore a natural choice to base DC on the interval logics proposed in [Dut95a, HMM83, Ven90], since these logics are logics for functions of time intervals.

2. Interval Logic

In this section we give the syntax, semantics, and proof system for Interval Logic (IL) based on [Dut95b, Dut95a]. Section 3 shows how DC extends IL.

2.1. Syntax

The formulas of IL are constructed from the following sets of symbols:

GVar: An infinite set of global *variables* $x, y, z, \ldots$. These variables are called global since their meaning is independent of time and time intervals.

TVar: An infinite set of *temporal variables* $v, v', \ldots$. The meaning of a temporal variable will be a real-valued interval function.

FSymb: An infinite set of global *function symbols* $f^n, g^m, \ldots$ equipped with arities $n, m \geqslant 0$. If f^n has arity $n = 0$ then f is called a *constant*. The meaning of a global function symbol f^n, $n > 0$, will be an n-ary function on real numbers which will be independent of time and time intervals.

RSymb: An infinite set of global *relation symbols* G^n, H^m equipped with arities $n, m \geqslant 0$. The meaning of a global relation symbol G^n, $n > 0$, will be an n-ary Boolean valued function on real numbers which will be independent of time and time intervals. The Boolean constants true and false are the only two global relation symbols with arity 0.

PLetter: An infinite set of temporal *propositional letters* $X, Y, \ldots$. The meaning of each temporal propositional letter will be a Boolean valued interval function.

The set of *terms* $\theta, \theta_i \in Terms$ is defined by the following abstract syntax:

$$\theta \quad ::= \quad x \mid \ell \mid v \mid f^n(\theta_1, \ldots, \theta_n)$$

where ℓ is a special symbol for an interval function denoting the interval length.

The set of *formulas* $\phi, \psi \in Formulas$ is defined by the following abstract syntax:

$$\phi \quad ::= \quad X \mid G^n(\theta_1, \ldots, \theta_n) \mid \neg\phi \mid \phi \vee \psi \mid \phi^\frown\psi \mid (\exists x)\phi$$

where $^\frown$ is a binary modality for "chopping" an interval into two consecutive sub-intervals. We also use φ, ϕ_i, ψ_i, and φ_i to denote formulas.

We will use standard notation for constants, e.g. 0,1, true and false, and for function and relation symbols of real arithmetic, e.g. $+$ and $\geqslant$.

2.2. Semantics

The meaning of terms and formulas are explained in this section. To do so one must define the meaning of global and temporal variables, (global) function and relation symbols, and (temporal) propositional letters.

We are only interested in functions and relations of real arithmetic, so let us assume that a total function $\underline{f}_i^n \in \mathbb{R}^n \to \mathbb{R}$ is associated with each n-ary function symbol f_i^n, and a total function $\underline{G}_i^n \in \mathbb{R}^n \to \{\mathrm{tt}, \mathrm{ff}\}$ is associated with each n-ary relation symbol G_i^n. In particular tt and ff are associated with true and false, respectively. We assume that $+, -, \ldots$ and $=, \leqslant, \ldots$ have their standard meaning.

The meaning of global variables is given by a *value assignment*, which is a function associating a real number with each global variable:

$$\mathcal{V} \in GVar \to \mathbb{R}$$

Let *Val* be the set of all value assignments.

The meaning of temporal variables and propositional letters, i.e. the "interval dependent symbols", is given by an interpretation:

$$\mathcal{J} \in \begin{pmatrix} TVar \\ \cup \\ PLetters \end{pmatrix} \to \begin{pmatrix} \mathbb{I}\text{ntv} \to \mathbb{R} \\ \cup \\ \mathbb{I}\text{ntv} \to \{\text{tt,ff}\} \end{pmatrix}$$

where
$$\mathbb{I}\text{ntv} \mathrel{\hat{=}} \{[b,e] \mid b,e \in \mathbb{R} \text{ and } b \leqslant e\},$$
$$\mathcal{J}(v)([b,e]) \in \mathbb{R} \text{ and } \mathcal{J}(X)([b,e]) \in \{\text{tt,ff}\}$$

associating a real-valued interval function with each temporal variable and a Boolean valued interval function with each temporal propositional letter. We will use the following abbreviations:

$$v_{\mathcal{J}} \mathrel{\hat{=}} \mathcal{J}(v) \quad \text{and} \quad X_{\mathcal{J}} \mathrel{\hat{=}} \mathcal{J}(X)$$

The *semantics of a term* θ in an interpretation $\mathcal{J}$ is a function

$$\mathcal{J}[\![\theta]\!] \in Val \times \mathbb{I}\text{ntv} \to \mathbb{R}$$

defined inductively on the structure of terms by:

$$\begin{aligned}
\mathcal{J}[\![x]\!](\mathcal{V},[b,e]) &= \mathcal{V}(x) \\
\mathcal{J}[\![\ell]\!]\ (\mathcal{V},[b,e]) &= e-b \\
\mathcal{J}[\![v]\!](\mathcal{V},[b,e]) &= v_{\mathcal{J}}([b,e]) \\
\mathcal{J}[\![f^n(\theta_1,\ldots,\theta_n)]\!]\ (\mathcal{V},[b,e]) &= \underline{f}^n(c_1,\ldots,c_n)
\end{aligned}$$

where $c_i = \mathcal{J}[\![\theta_i]\!]\ (\mathcal{V},[b,e])$, for $1 \leqslant i \leqslant n$.

The *semantics of a formula* ϕ in an interpretation $\mathcal{J}$ is a function

$$\mathcal{J}[\![\phi]\!] \in Val \times \mathbb{I}\text{ntv} \to \{\text{tt,ff}\}$$

defined inductively on the structure of formulas below, where the following abbreviations will be used:

$$\begin{aligned}
\mathcal{J},\mathcal{V},[b,e] \models \phi &\mathrel{\hat{=}} \mathcal{J}[\![\phi]\!]\ (\mathcal{V},[b,e]) = \text{tt} \\
\mathcal{J},\mathcal{V},[b,e] \not\models \phi &\mathrel{\hat{=}} \mathcal{J}[\![\phi]\!]\ (\mathcal{V},[b,e]) = \text{ff}
\end{aligned}$$

The definition of $\mathcal{J}[\![\phi]\!]$ is:

1) $\mathcal{J},\mathcal{V},[b,e] \models X$ iff $X_{\mathcal{J}}([b,e]) = \mathrm{tt}$

2) $\mathcal{J},\mathcal{V},[b,e] \models G^n(\theta_1,\ldots,\theta_n)$
iff $\underline{G}^n(\mathcal{J}[\![\theta_1]\!](\mathcal{V},[b,e]),\ldots,\mathcal{J}[\![\theta_n]\!](\mathcal{V},[b,e])) = \mathrm{tt}$

3) $\mathcal{J},\mathcal{V},[b,e] \models \neg\phi$ iff $\mathcal{J},\mathcal{V},[b,e] \not\models \phi$

4) $\mathcal{J},\mathcal{V},[b,e] \models \phi \vee \psi$ iff $\mathcal{J},\mathcal{V},[b,e] \models \phi$ or $\mathcal{J},\mathcal{V},[b,e] \models \psi$

5) $\mathcal{J},\mathcal{V},[b,e] \models \phi^\frown\psi$
iff $\mathcal{J},\mathcal{V},[b,m] \models \phi$ and $\mathcal{J},\mathcal{V},[m,e] \models \psi$, for some $m \in [b,e]$

6) $\mathcal{J},\mathcal{V},[b,e] \models (\exists x)\phi$
iff $\mathcal{J},\mathcal{V}',[b,e] \models \phi$ $\left\{\begin{array}{l}\text{for some value assignment } \mathcal{V}'\\ \text{which is } x\text{-equivalent to } \mathcal{V}\end{array}\right\}$
where $\mathcal{V}$ and $\mathcal{V}'$ are called *x-equivalent* iff $\mathcal{V}(y) = \mathcal{V}'(y)$
for any global variable y which is different from x.

A formula ϕ is *valid*, written $\models \phi$, iff $\mathcal{J},\mathcal{V},[b,e] \models \phi$, for any interpretation $\mathcal{J}$, value assignment $\mathcal{V}$, and interval $[b,e]$. A formula ψ is *satisfiable* iff $\mathcal{J},\mathcal{V},[b,e] \models \psi$ for some interpretation $\mathcal{J}$, value assignment $\mathcal{V}$, and interval $[b,e]$.

2.2.1. Abbreviations and Conventions

The following abbreviations will be used:

$\Diamond\phi$	$\widehat{=}$	$\mathrm{true}^\frown(\phi^\frown\mathrm{true})$	reads: "for some sub-interval: ϕ"
$\Box\phi$	$\widehat{=}$	$\neg\Diamond(\neg\phi)$	reads: "for all sub-intervals: ϕ"

Furthermore, the standard abbreviations from predicate logic will be used e.g.

$\phi \wedge \psi$	$\widehat{=}$	$\neg((\neg\phi) \vee (\neg\psi))$
$\phi \Rightarrow \psi$	$\widehat{=}$	$(\neg\phi) \vee \psi$
$\phi \Leftrightarrow \psi$	$\widehat{=}$	$(\phi \Rightarrow \psi) \wedge (\psi \Rightarrow \phi)$
$(\forall x)\phi$	$\widehat{=}$	$\neg((\exists x)\neg\phi)$

When $\neg, (\exists x), (\forall x), \Box$, and $\Diamond$ occur in formulas they have higher precedence than the binary connectives and the modality $^\frown$, e.g. $(\Box\phi) \Rightarrow (((\forall x)(\neg\psi))^\frown\varphi)$ can be written as $\Box\phi \Rightarrow ((\forall x)\neg\psi^\frown\varphi)$.

2.3. Proof System

The proof system (or calculus) of IL we adopt is called S' in [Dut95b]. To formulate the axioms and inference rules, we need the standard notion of free (global) variables. Moreover, a term (formula) is called *flexible* if a temporal variable, the symbol ℓ or a propositional letter occurs in the term (formula). A term or formula which is not flexible is also called *rigid*.

The axioms of IL are:

A0 : $\ell \geqslant 0$

A1 : $((\phi^\frown\psi) \wedge \neg(\phi^\frown\varphi)) \Rightarrow (\phi^\frown(\psi \wedge \neg\varphi))$
$((\phi^\frown\psi) \wedge \neg(\varphi^\frown\psi)) \Rightarrow ((\phi \wedge \neg\varphi)^\frown\psi)$

A2 : $((\phi^\frown\psi)^\frown\varphi) \Leftrightarrow (\phi^\frown(\psi^\frown\varphi))$

R : $(\phi^\frown\psi) \Rightarrow \phi$ if ϕ is a rigid formula
$(\phi^\frown\psi) \Rightarrow \psi$ if ψ is a rigid formula

B : $((\exists x)\phi^\frown\psi) \Rightarrow (\exists x)(\phi^\frown\psi)$ if x is not free in ψ
$(\phi^\frown(\exists x)\psi) \Rightarrow (\exists x)(\phi^\frown\psi)$ if x is not free in ϕ

L1 : $((\ell = x)^\frown\phi) \Rightarrow \neg((\ell = x)^\frown\neg\phi)$
$(\phi^\frown(\ell = x)) \Rightarrow \neg(\neg\phi^\frown(\ell = x))$

L2 : $(x \geqslant 0 \wedge y \geqslant 0) \Rightarrow ((\ell = x + y) \Leftrightarrow ((\ell = x)^\frown(\ell = y)))$

L3 : $\phi \Rightarrow (\phi^\frown(\ell = 0))$

L4 : $\phi \Rightarrow ((\ell = 0)^\frown\phi)$

The inference rules of IL are:

MP : if ϕ and $\phi \Rightarrow \psi$ then ψ (modus ponens)
G : if ϕ then $(\forall x)\phi$ (generalization)
N1 : if ϕ then $\neg(\neg\phi^\frown\psi)$
N2 : if ϕ then $\neg(\psi^\frown\neg\phi)$
M1 : if $\phi \Rightarrow \psi$ then $(\phi^\frown\varphi) \Rightarrow (\psi^\frown\varphi)$
M2 : if $\phi \Rightarrow \psi$ then $(\varphi^\frown\phi) \Rightarrow (\varphi^\frown\psi)$

The inference rules N1 and N2 are called rules of necessitation, and the inference rules M1 and M2 are the monotony rules for chop. The inference rule G is the standard generalization rule from first order logic.

The proof system also contains axioms of first order predicate logic with equality. Any axiomatic basis can be chosen, and we will use "PL" when we refer to predicate logic axioms, theorems, and inference rules. Special care must, however, be taken when universally quantified formulas are instantiated:

To formulate an axiom schema for universal quantification we define: A term θ is called *free for* x in ϕ if x does not occur freely in ϕ within a scope of $\exists y$ or $\forall y$, where y is any variable occurring in θ.

For example, y is free for x in $(\exists z)(z > x)$; whereas y is not free for x in $(\exists y)(y > x)$. Note that $(\forall x)(\exists z)(z > x)$ and $(\forall x)(\exists y)(y > x)$ are both valid. Instantiation of x with y in the first formula yields $(\exists z)(z > y)$, which is a valid formula. However, instantiation of x with y in the second formula yields $(\exists y)(y > y)$, which is not valid.

A formula is called *chop free* if $^\frown$ does not occur in the formula.

Consider for example the following universally quantified and valid formula: $(\forall x)(((\ell = x)^\frown(\ell = x)) \Rightarrow (\ell = 2x))$. Instantiating this formula, which is not chop free, with the term ℓ, which is flexible, yields $((\ell = \ell)^\frown(\ell = \ell)) \Rightarrow (\ell = 2\ell)$, which is not valid.

Therefore, a side-condition occurs in the following axiom schema:

$$\mathrm{Q} : (\forall x)\phi(x) \Rightarrow \phi(\theta) \left(\begin{array}{l} \text{if either } \theta \text{ is free for } x \text{ in } \phi(x) \text{ and } \theta \text{ is rigid} \\ \text{or } \theta \text{ is free for } x \text{ in } \phi(x) \text{ and } \phi(x) \text{ is chop free.} \end{array} \right)$$

A *proof* of ϕ is a finite sequence of formulas $\phi_1 \cdots \phi_n$, where ϕ_n is ϕ, and each ϕ_i is either an instance of one of the above axiom schemas or obtained by applying one of the above inference rules to previous members of the sequence. We write $\vdash_{il} \phi$ to denote that there exists a proof of ϕ in IL and ϕ is called a *theorem* of IL.

The proof system is *sound*, i.e. if $\vdash_{il} \phi$ then $\models \phi$ [Dut95b].

A *deduction of ϕ in IL from a set of formulas* Γ is a sequence of formulas $\phi_1 \cdots \phi_n$, where ϕ_n is ϕ, and each ϕ_i is either a member of Γ, an instance of one of the above axiom schemas or obtained by applying one of the above inference rules to previous members of the sequence. We write $\Gamma \vdash_{il} \phi$ to denote that there exists a deduction of ϕ from Γ in IL, and we write $\Gamma, \phi \vdash_{il} \psi$ for $(\Gamma \cup \{\phi\}) \vdash_{il} \psi$.

IL is an extension of the modal logic S4, e.g. [HuC68], since the following three theorems and one deduction can be proven in IL (remember that $\Box\phi$ is an abbreviation for $\neg\Diamond\neg\phi$ and that $\Diamond\psi$ is an abbreviation of $\mathrm{true}^\frown(\psi^\frown\mathrm{true})$):

T1	$\Box(\phi \Rightarrow \psi) \Rightarrow (\Box\phi \Rightarrow \Box\psi)$
T2	$\Box\phi \Rightarrow \phi$
T3	$\Box\phi \Rightarrow \Box\Box\phi$
R4	$\phi \vdash_{il} \Box\phi$

Proof. We only give proof for R4:

1.	ϕ	assumption
2.	$\neg(\neg\phi^\frown\mathrm{true})$	1., N1
3.	$\neg(\mathrm{true}^\frown\neg(\neg(\neg\phi^\frown\mathrm{true})))$	2., N2
4.	$(\neg\phi^\frown\mathrm{true}) \Rightarrow \neg(\neg(\neg\phi^\frown\mathrm{true}))$	PL
5.	$(\mathrm{true}^\frown(\neg\phi^\frown\mathrm{true})) \Rightarrow (\mathrm{true}^\frown\neg(\neg(\neg\phi^\frown\mathrm{true})))$	4., M2
6.	$\neg(\mathrm{true}^\frown\neg(\neg(\neg\phi^\frown\mathrm{true}))) \Rightarrow \neg(\mathrm{true}^\frown(\neg\phi^\frown\mathrm{true}))$	5., PL
7.	$\neg(\mathrm{true}^\frown(\neg\phi^\frown\mathrm{true}))$	3., 6., MP

□

The following theorems and deduction will be used later in the proof of the deduction theorem:

T5	$\Box\phi \Rightarrow \neg(\neg\phi^\frown\psi)$
R6	$\Box\phi \Rightarrow \psi \vdash_{il} \Box\phi \Rightarrow \Box\psi$
T7	$\Box(\phi_1 \Rightarrow \phi_2) \Rightarrow ((\phi_1^\frown\psi) \Rightarrow (\phi_2^\frown\psi))$

Proof. We prove $(\neg\phi^\frown\psi) \Rightarrow \neg\Box\phi$, i.e. $(\neg\phi^\frown\psi) \Rightarrow (\mathrm{true}^\frown(\neg\phi^\frown\mathrm{true}))$, to prove T5:

1.	$\psi \Rightarrow \mathrm{true}$	PL
2.	$(\neg\phi^\frown\psi) \Rightarrow (\neg\phi^\frown\mathrm{true})$	1., M2
3.	$\ell = 0 \Rightarrow \mathrm{true}$	PL
4.	$(\ell = 0^\frown(\neg\phi^\frown\psi)) \Rightarrow (\mathrm{true}^\frown(\neg\phi^\frown\psi))$	3., M1
5.	$(\mathrm{true}^\frown(\neg\phi^\frown\psi)) \Rightarrow (\mathrm{true}^\frown(\neg\phi^\frown\mathrm{true}))$	2., M2
6.	$(\ell = 0^\frown(\neg\phi^\frown\psi)) \Rightarrow (\mathrm{true}^\frown(\neg\phi^\frown\mathrm{true}))$	4., 5., PL
7.	$(\neg\phi^\frown\psi) \Rightarrow (\ell = 0^\frown(\neg\phi^\frown\psi))$	L4
8.	$(\neg\phi^\frown\psi) \Rightarrow (\mathrm{true}^\frown(\neg\phi^\frown\mathrm{true}))$	7., 6., PL

The proof of R6 is given by:

1.	$\Box\phi \Rightarrow \psi$	assumption
2.	$\Box(\Box\phi \Rightarrow \psi)$	1., R4
3.	$\Box(\Box\phi \Rightarrow \psi) \Rightarrow (\Box\Box\phi \Rightarrow \Box\psi)$	T1
4.	$\Box\Box\phi \Rightarrow \Box\psi$	2., 3., MP
5.	$\Box\phi \Rightarrow \Box\Box\phi$	T3
6.	$\Box\phi \Rightarrow \Box\psi$	5., 4., PL

The proof of T7 is left for the reader. □

The deduction theorem for IL is

Theorem 2.1. (Deduction) If a deduction, $\Gamma, \phi \vdash_{il} \psi$, involves no application of the generalization rule G of which the quantified variable is free in ϕ, then $\Gamma \vdash_{il} \Box\phi \Rightarrow \psi$.

Proof. The proof is by induction on the length n of the deduction $\Gamma, \phi \vdash_{il} \psi$.

Base step: $n = 1$. Then ψ must either be ϕ, a member of Γ, or an axiom.

Case ψ is ϕ: This case is simple since $\vdash_{il} \Box\phi \Rightarrow \phi$ by T2 and thus trivially $\Gamma \vdash_{il} \Box\phi \Rightarrow \psi$.

Case ψ is an axiom or a member of Γ: In this case the following deduction establishes $\Gamma \vdash_{il} \Box\phi \Rightarrow \psi$:

1.	ψ	
2.	$\psi \Rightarrow (\Box\phi \Rightarrow \psi)$	PL
3.	$\Box\phi \Rightarrow \psi$	1., 2., MP

Inductive step: Suppose $n > 0$. The induction hypothesis is: If $\Gamma \cup \{\phi\} \vdash_{il} \varphi$ by a deduction of length shorter than n which does not contain an application of the generalization rule G of which the quantified variable is free in ϕ, then $\Gamma \vdash_{il} \Box\phi \Rightarrow \varphi$.

The case where ψ is either ϕ, a member of Γ, or an axiom is as above. Otherwise an inference rule is applied in the last step in the deduction:

Case MP: The deduction from $\Gamma \cup \{\phi\}$ has the form:

$$\begin{array}{l} \vdots \\ \psi_1 \\ \vdots \\ \psi_1 \Rightarrow \psi \\ \vdots \\ \psi \end{array}$$

There are deductions of $\Box\phi \Rightarrow \psi_1$ and $\Box\phi \Rightarrow (\psi_1 \Rightarrow \psi)$ from Γ by the induction hypothesis. A deduction of $\Box\phi \Rightarrow \psi$ from Γ can be given as:

$$\left.\begin{array}{ll} \vdots & \\ k. & \Box\phi \Rightarrow \psi_1 \end{array}\right\} \text{ deduction of } \Box\phi \Rightarrow \psi_1 \text{ from } \Gamma$$

$$\left.\begin{array}{ll} \vdots & \\ l. & \Box\phi \Rightarrow (\psi_1 \Rightarrow \psi) \end{array}\right\} \text{ deduction of } \Box\phi \Rightarrow (\psi_1 \Rightarrow \psi) \text{ from } \Gamma$$

$$\begin{array}{lll} l+1. & (\Box\phi \Rightarrow (\psi_1 \Rightarrow \psi)) \Rightarrow ((\Box\phi \Rightarrow \psi_1) \Rightarrow (\Box\phi \Rightarrow \psi)) & \text{PL} \\ l+2. & (\Box\phi \Rightarrow \psi_1) \Rightarrow (\Box\phi \Rightarrow \psi) & l, l+1, \text{MP} \\ l+3. & \Box\phi \Rightarrow \psi & k., l+2, \text{MP} \end{array}$$

Case G: ψ has the form $(\forall x)\psi_1$, and the deduction from $\Gamma \cup \{\phi\}$ has the form:

$$\begin{array}{l} \vdots \\ \psi_1 \\ \vdots \\ (\forall x)\psi_1 \end{array}$$

Note that x does not occur freely in ϕ and hence in $\Box\phi$. Thus, we have from PL:

$$\vdash_{il} (\forall x)(\Box\phi \Rightarrow \psi_1) \Rightarrow (\Box\phi \Rightarrow (\forall x)\psi_1)$$

By the induction hypothesis there is a deduction of $\Box\phi \Rightarrow \psi_1$ from Γ. A deduction of $\Box\phi \Rightarrow (\forall x)\psi_1$ from Γ can be given as:

$$\left.\begin{array}{ll} \vdots & \\ k. & \Box\phi \Rightarrow \psi_1 \end{array}\right\} \text{ deduction of } \Box\phi \Rightarrow \psi_1 \text{ from } \Gamma$$

$$\begin{array}{lll} k+1. & (\forall x)(\Box\phi \Rightarrow \psi_1) & k, \text{G} \\ k+2. & (\forall x)(\Box\phi \Rightarrow \psi_1) \Rightarrow (\Box\phi \Rightarrow (\forall x)\psi_1) & \text{PL} \\ k+3. & \Box\phi \Rightarrow (\forall x)\psi_1 & k+1., k+2., \text{MP} \end{array}$$

Case N1: ψ has the form $\neg(\neg\psi_1 \widehat{\ } \varphi)$, and the deduction from $\Gamma \cup \{\phi\}$ has the form:

$$\begin{array}{l} \vdots \\ \psi_1 \\ \vdots \\ \neg(\neg\psi_1 \widehat{\ } \varphi) \end{array}$$

By the induction hypothesis there is a deduction of $\Box\phi \Rightarrow \psi_1$ from Γ. A deduction of $\Box\phi \Rightarrow \neg(\neg\psi_1 \widehat{\ } \varphi)$ from Γ can be given as:

$$\left.\begin{array}{ll} \vdots & \\ k. & \Box\phi \Rightarrow \psi_1 \end{array}\right\} \text{ deduction of } \Box\phi \Rightarrow \psi_1 \text{ from } \Gamma$$

$$\begin{array}{lll} k+1. & \Box\phi \Rightarrow \Box\psi_1 & k., \text{R6} \\ k+2. & \Box\psi_1 \Rightarrow \neg(\neg\psi_1 \widehat{\ } \varphi) & \text{T5} \\ k+3. & \Box\phi \Rightarrow \neg(\neg\psi_1 \widehat{\ } \varphi) & k+1., k+2., \text{PL} \end{array}$$

Case N2: Similar to N1.

Case M1: ψ has the form $(\psi_1{}^\frown\varphi) \Rightarrow (\psi_2{}^\frown\varphi)$, and the deduction from $\Gamma \cup \{\phi\}$ has the form:

$$\vdots$$
$$\psi_1 \Rightarrow \psi_2$$
$$\vdots$$
$$(\psi_1{}^\frown\varphi) \Rightarrow (\psi_2{}^\frown\varphi)$$

By the induction hypothesis there is a deduction of $\Box\phi \Rightarrow (\psi_1 \Rightarrow \psi_2)$ from Γ. A deduction of $\Box\phi \Rightarrow ((\psi_1{}^\frown\varphi) \Rightarrow (\psi_2{}^\frown\varphi))$ from Γ can be given as:

$$\left.\begin{array}{ll} \vdots & \\ k. & \Box\phi \Rightarrow (\psi_1 \Rightarrow \psi_2) \end{array}\right\} \text{ deduction of } \Box\phi \Rightarrow (\psi_1 \Rightarrow \psi_2) \text{ from } \Gamma$$

$$\begin{array}{lll} k+1. & \Box\phi \Rightarrow \Box(\psi_1 \Rightarrow \psi_2) & k., \text{R6} \\ k+2. & \Box(\psi_1 \Rightarrow \psi_2) \Rightarrow ((\psi_1{}^\frown\varphi) \Rightarrow (\psi_2{}^\frown\varphi)) & \text{T7} \\ k+3. & \Box\phi \Rightarrow ((\psi_1{}^\frown\varphi) \Rightarrow (\psi_2{}^\frown\varphi)) & k+1., k+2., \text{PL} \end{array}$$

Case M2: Similar to M1. This ends the proof of the deduction theorem. □

Proofs can sometimes be obtained more easily by using the deduction theorem. We can, for example, prove

T8 $\quad \Box(\phi \Rightarrow \psi) \Rightarrow \Box(\Box\phi \Rightarrow \Box\psi)$

from a deduction of $\Box(\Box\phi \Rightarrow \Box\psi)$ from $\{(\phi \Rightarrow \psi)\}$ using Theorem 2.1:

1. $\phi \Rightarrow \psi$
2. $\Box(\phi \Rightarrow \psi)$ $\quad$ 1, R4
3. $\Box\phi \Rightarrow \Box\psi$ $\quad$ 2, T1, MP
4. $\Box(\Box\phi \Rightarrow \Box\psi)$ $\quad$ 3, R4

Remark: It can be proved that the proof system of IL [Dut95b] and the proof system for the interval logic in [ZhH96a] are complete wrt. value/time domain satisfying axioms for totally ordered commutative groups. Real numbers are a totally ordered commutative group. Unfortunately, it is impossible to establish a first order logic having real numbers as its only model. □

3. Duration Calculus

3.1. Syntax

In this section we establish DC as an extension of IL in the sense that temporal variables $v \in TVar$ have a structure:

$$\int S$$

where S is called a *state expression* and is generated from a set $SVar$ of *state variables* $P, Q, R, \ldots$, according to the following abstract syntax:

$$S \quad ::= \quad 0 \mid 1 \mid P \mid \neg S_1 \mid S_1 \vee S_2$$

We will use the same abbreviations for propositional connectives in state expressions as introduced for IL formulas.

Remark: The propositional connectives $\neg$ and $\vee$ occur both in state expressions and in formulas but, as we shall see below, with different semantics. This does not give problems as state expressions always occur in the context of $\int$. □

3.2. Semantics

When we generate temporal variables from state variables, the semantics of temporal variables must be derived from the semantics of the state variables. To this end we introduce an *interpretation for state variables* (and propositional letters) as a function:

$$\mathcal{I} \in \begin{pmatrix} SVar \\ \cup \\ PLetters \end{pmatrix} \to \begin{pmatrix} \mathbb{T}\text{ime} \to \{0,1\} \\ \cup \\ \mathbb{I}\text{ntv} \to \{\text{tt},\text{ff}\} \end{pmatrix}$$

where $\mathcal{I}(P) \in \mathbb{T}\text{ime} \to \{0,1\}$, $\mathcal{I}(X) \in \mathbb{I}\text{ntv} \to \{\text{tt},\text{ff}\}$, and each function $\mathcal{I}(P)$ has at most a finite number of discontinuity points in any interval $[b,e]$, hence $\mathcal{I}(P)$ is integrable in any bounded interval.

The semantics of a state expression S, given an interpretation $\mathcal{I}$, is a function:

$$\mathcal{I}[\![S]\!] \in \mathbb{T}\text{ime} \to \{0,1\}$$

defined inductively on the structure of state expressions by:

$$\begin{array}{lcl} \mathcal{I}[\![0]\!](t) & = & 0 \\ \mathcal{I}[\![1]\!](t) & = & 1 \\ \mathcal{I}[\![P]\!](t) & = & \mathcal{I}(P)(t) \\ \mathcal{I}[\![(\neg S)]\!](t) & = & 1 - \mathcal{I}[\![S]\!](t) \\ \mathcal{I}[\![(S_1 \vee S_2)]\!](t) & = & \begin{cases} 0 & \text{if } \mathcal{I}[\![S_1]\!](t) = 0 \text{ and } \mathcal{I}[\![S_2]\!](t) = 0 \\ 1 & \text{otherwise} \end{cases} \end{array}$$

We shall use the abbreviation $S_{\mathcal{I}} \mathrel{\widehat{=}} \mathcal{I}[\![S]\!]$. We see by this semantics that each function $S_{\mathcal{I}}$ has at most a finite number of discontinuity points in any interval $[b,e]$ and is thus integrable in any bounded interval.

The semantics of temporal variables, which now has the form $\int S$, is given by a function $\mathcal{I}[\![\int S]\!] \in \mathbb{I}\text{ntv} \to \mathbb{R}$ defined by:

$$\mathcal{I}[\![\int S]\!][b,e] \quad = \quad \int_b^e S_{\mathcal{I}}(t)dt$$

This function can be used to induce an interpretation $\mathcal{J}_{\mathcal{I}}$ for temporal variables v of the form $\int S$ and propositional letters from $\mathcal{I}$:

$$\begin{array}{lcll} \mathcal{J}_{\mathcal{I}}(X) & = & \mathcal{I}(X) & \text{for any propositional letter } X \\ \mathcal{J}_{\mathcal{I}}(v) & = & \mathcal{I}[\![\int S]\!] & \text{when } v \text{ is } \int S \end{array}$$

The semantics of a duration calculus formula ϕ, given an interpretation $\mathcal{I}$ to state variables, is a function:

$$\mathcal{I}[\![\phi]\!] \in Val \times \mathbb{I}\text{ntv} \to \{\text{tt},\text{ff}\}$$

for which we use the abbreviations:

$$\begin{array}{lcl} \mathcal{I},\mathcal{V},[b,e] \models_{dc} \phi & \mathrel{\widehat{=}} & \mathcal{I}[\![\phi]\!]\,(\mathcal{V},[b,e]) = \text{tt} \\ \mathcal{I},\mathcal{V},[b,e] \not\models_{dc} \phi & \mathrel{\widehat{=}} & \mathcal{I}[\![\phi]\!]\,(\mathcal{V},[b,e]) = \text{ff} \end{array}$$

The function can be defined as follows:

$$\mathcal{I},\mathcal{V},[b,e] \models_{dc} \phi \quad \text{iff} \quad \mathcal{J}_{\mathcal{I}},\mathcal{V},[b,e] \models \phi$$

The notions of satisfiability and validity of DC formulas are defined as for IL formulas.

Remark: For two given interpretations $\mathcal{I}$ and $\mathcal{I}'$ whose values for any state variable P and interval $[b,e]$ disagree for at most a finite number of points in $[b,e]$ we have

$$\mathcal{I}[\![\int P]\!]\,[b,e] \;=\; \mathcal{I}'[\![\int P]\!]\,[b,e]$$

No formula can distinguish $\mathcal{I}$ and $\mathcal{I}'$ since state expressions only occur within the context of $\int$. We can therefore define that $\mathcal{I}$ and $\mathcal{I}'$ are equivalent, and build equivalence classes of interpretations which no formula can distinguish. Such an equivalence class will contain an interpretation which for any state variable P is, say left, continuous, and for any interval $[b,e]$ has no more discontinuity points than any other interpretation in the equivalence class. This "minimal" left continuous function could be taken as a representative of the equivalence class, and in this sense the interpretation of a state variables is a stable, say left continuous, function. C.f. discussions in Section 1.2. □

The following abbreviation will be used frequently:

$$\lceil\rceil \;\widehat{=}\; \ell = 0$$
$$\lceil S\rceil \;\widehat{=}\; \int S = \ell \wedge \ell > 0$$

The formula $\lceil S\rceil$ holds in an interval $[b,e]$ iff $b < e$ and S is 1 everywhere (almost) in $[b,e]$, i.e. S may be 0 at at most a finite number of time points in $[b,e]$.

Gas Burner: The requirement of the gas burner can be formalized in DC by:

$$\ell \geqslant 60 \;\Rightarrow\; 20\int \mathrm{Leak} \leqslant \ell$$

and the two design decisions can be formalized in DC by

$$\Box(\lceil \mathrm{Leak}\rceil \;\Rightarrow\; \ell \leqslant 1)$$

and[1]

$$\Box((\lceil \mathrm{Leak}\rceil ^\frown \lceil \neg \mathrm{Leak}\rceil ^\frown \lceil \mathrm{Leak}\rceil) \;\Rightarrow\; \ell \geqslant 30)$$

□

3.3. Proof System

Since DC is an extension of IL we adopt all axioms and inference rules of IL from the previous section as axioms and inference rules for DC. Furthermore, we add axioms reflecting the structure which DC adds to temporal variables:

(DC-A1) $\int 0 = 0$

(DC-A2) $\int 1 = \ell$

(DC-A3) $\int S \geqslant 0$

(DC-A4) $\int S_1 + \int S_2 = \int(S_1 \vee S_2) + \int(S_1 \wedge S_2)$

(DC-A5) $((\int S = x)^\frown(\int S = y)) \Rightarrow (\int S = x + y)$

(DC-A6) $\int S_1 = \int S_2$, provided $S_1 \Leftrightarrow S_2$ holds in propositional logic

[1] It can be proved that this formula defines the state of Leak which satisfies the mathematical formulation of the second design decision for the Gas Burner on page 286.

Furthermore, we add two induction rules:

IR1: Let $H(X)$ be a formula possibly containing the propositional letter X, and let S be any state expression.

$$\begin{array}{ll} \text{If} & H(\lceil\rceil) \text{ and } H(X) \Rightarrow H(X \vee (X^\frown\lceil S\rceil) \vee (X^\frown\lceil\neg S\rceil)) \\ \text{then} & H(\text{true}) \end{array}$$

where $H(\phi)$ denotes the formula obtained from $H(X)$ by replacing every occurrence of X in H with ϕ.

IR2: Let $H(X)$ be a formula possibly containing the propositional letter X, and let S be any state expression.

$$\begin{array}{ll} \text{If} & H(\lceil\rceil) \text{ and } H(X) \Rightarrow H(X \vee (\lceil S\rceil^\frown X) \vee (\lceil\neg S\rceil^\frown X)) \\ \text{then} & H(\text{true}) \end{array}$$

In these rules $H(\lceil\rceil)$ is called the *base case*, and $H(X)$ is called the *induction hypothesis.*

Remark: The soundness of these two induction rules relies on the finite variability property of functions $S_{\mathcal{I}}$ (see below). Furthermore, in the relative completeness proof (Lemma 4.2) we shall see that the induction rules have a major rôle in the formalization of the finite variability property. □

A *proof* of ϕ in DC is a finite sequence of formulas $\phi_1 \cdots \phi_n$, where ϕ_n is ϕ, and each ϕ_i is either an instance of one of the above axiom schemas or an axiom schema of IL or obtained by applying one of the induction rules or the inference rules of IL to previous members of the sequence. We write $\vdash_{dc} \phi$ to denote that there exists a proof of ϕ in DC and ϕ is called a *theorem* of DC. Deduction in DC is defined similarly to deduction in IL and by $\Gamma \vdash_{dc} \phi$ we denote that there exists a deduction of ϕ in DC from Γ.

The definitions and lemmas below are convenient for the soundness proof.

Definition. (Equivalence) Given an interval $[b,e]$ and an interpretation $\mathcal{I}$. We call two formulas ϕ and ψ *equivalent in* $[b,e]$ *of* $\mathcal{I}$ if

$$\mathcal{I}, \mathcal{V}, [c,d] \models_{dc} \phi \text{ iff } \mathcal{I}, \mathcal{V}, [c,d] \models_{dc} \psi$$

for any value assignment $\mathcal{V}$ and interval $[c,d]$ where $[c,d] \subseteq [b,e]$.

Definition. (Finite alternation) Given a state expression S. The formula $FA^i(S)$, for $i \geqslant 0$, describes less than i alternations of S:

$$\begin{array}{lcl} FA^0(S) & \widehat{=} & \lceil\rceil \\ FA^{i+1}(S) & \widehat{=} & FA^i(S) \vee (\lceil S\rceil^\frown FA^i(S)) \vee (\lceil\neg S\rceil^\frown FA^i(S)) \end{array}$$

Lemma 3.1. (Equivalence) For a given state expression S, interval $[b,e]$, and interpretation $\mathcal{I}$, there is a natural number n so that true and $FA^n(S)$ are equivalent in $[b,e]$ of $\mathcal{I}$.

Proof. Follows since $S_{\mathcal{I}}$ has at most a finite number of alternations in $[b,e]$. □

Lemma 3.2. (Substitution) Let $\varphi(X)$ be a formula in which the propositional letter X may occur, let $[b,e]$ be an interval, and let $\mathcal{I}$ be an interpretation. Then for any two formulas ϕ_1 and ϕ_2:

If ϕ_1 and ϕ_2 are equivalent in $[b,e]$ of $\mathcal{I}$
then $\varphi(\phi_1)$ and $\varphi(\phi_2)$ are equivalent in $[b,e]$ of $\mathcal{I}$

Proof. By structural induction on $\varphi(X)$. □

Lemma 3.3. If ψ does not contain free variables and X, then

$$\models_{dc} \phi(X) \text{ implies } \models_{dc} \phi(\psi)$$

Proof. Suppose $\models_{dc} \phi(X)$. Then for arbitrary $\mathcal{I}, \mathcal{V}, [a,b]$ we must show that $\mathcal{I}, \mathcal{V}, [a,b] \models_{dc} \phi(\psi)$. Define $\mathcal{I}'$ so that $\mathcal{I}'(X)[c,d] = \mathcal{I}[\![\psi]\!](\mathcal{V}, [c,d])$ which is independent of $\mathcal{V}$ since ψ contains no free variables. Otherwise $\mathcal{I}'$ is as $\mathcal{I}$. Since X does not occur in ψ and in $\phi(\psi)$: $\mathcal{I}, \mathcal{V}, [a,b] \models \phi(\psi)$ iff $\mathcal{I}', \mathcal{V}, [a,b] \models \phi(\psi)$. The formulas X and ψ are (by construction) equivalent in $[a,b]$ of $\mathcal{I}'$. Thus by the substitution lemma: $\mathcal{I}', \mathcal{V}, [a,b] \models_{dc} \phi(X)$ iff $\mathcal{I}', \mathcal{V}, [a,b] \models_{dc} \phi(\psi)$. Therefore $\mathcal{I}, \mathcal{V}, [a,b] \models_{dc} \phi(\psi)$. □

Theorem 3.1. (Soundness) The proof system of DC is sound, i.e.

$$\vdash_{dc} \phi \text{ implies } \models_{dc} \phi$$

Proof. To prove soundness, it suffices to prove that every axiom is valid and that every inference rule preserves validity, i.e. it yields a valid formula when applied to valid formulas. We only give the proof for IR2. The proof for IR1 is similar, and the proofs for the axioms and other inference rules are simple.

So suppose that

$$(I) \quad \models_{dc} H(\lceil\!\rceil), \text{ i.e. } \models_{dc} H(FA^0(S)), \text{ and}$$

$$(II) \quad \models_{dc} H(X) \Rightarrow H(X \vee (\lceil S \rceil^\frown X) \vee (\lceil \neg S \rceil^\frown X))$$

We must establish that $\models_{dc} H(\text{true})$. We first prove $\models_{dc} H(FA^n(S))$, for any natural number n, by induction on n.

The case for $n = 0$ is established by (I).

Inductive step: From Lemma 3.3 and (II) we get

$$\models_{dc} H(FA^n(S)) \Rightarrow H(FA^{n+1}(S))$$

Combining this with the induction hypothesis $\models_{dc} H(FA^n(S))$ we get

$$\models_{dc} H(FA^{n+1}(S))$$

To show $\models_{dc} H(\text{true})$, we must show that $\mathcal{I}, \mathcal{V}, [b,e] \models_{dc} H(\text{true})$ for any interpretation $\mathcal{I}$, value assignment $\mathcal{V}$, and interval $[b,e]$. But by the equivalence lemma there is a natural number k so that true and $FA^k(S)$ are equivalent in $[b,e]$ of $\mathcal{I}$, and by the substitution lemma $H(\text{true})$ and $H(FA^k(S))$ are equivalent in $[b,e]$ of $\mathcal{I}$ also.

Thus, since from above we have that

$$\mathcal{I}, \mathcal{V}, [b,e] \models H(FA^k(S))$$

then we also have that

$$\mathcal{I}, \mathcal{V}, [b,e] \models H(\text{true})$$

□

In order to simplify proofs in DC, we establish the deduction theorem:

Theorem 3.2. (Deduction)

$$\Gamma, \phi \vdash_{dc} \psi \text{ implies } \Gamma \vdash_{dc} \Box\phi \Rightarrow \psi$$

provided a deduction $\Gamma, \phi \vdash_{dc} \psi$ involves no application of the generalization rule G of which the quantified variable is free in ϕ and every application of the induction rules with hypothesis $H(X)$ satisfies that X does not occur in ϕ.

Proof. To the proof of the deduction theorem for IL we must add the cases where the induction rules are applied as the last step of the deduction. All other cases remain the same.

Case IR1: ψ has the form $H(\text{true})$, and the deduction from $\Gamma \cup \{\phi\}$ has the form:

$$\begin{array}{l}\vdots\\ H(\lceil\ \rceil)\\ \vdots\\ H(X) \Rightarrow H(X \vee (X^\frown\lceil S\rceil) \vee (X^\frown\lceil \neg S\rceil))\\ \vdots\\ H(\text{true})\end{array}$$

By the induction hypothesis there are deductions from Γ of $\Box\phi \Rightarrow H(\lceil\ \rceil)$ and $\Box\phi \Rightarrow (H(X) \Rightarrow H(X \vee (X^\frown\lceil S\rceil) \vee (X^\frown\lceil \neg S\rceil)))$. In the following deduction of $\Box\phi \Rightarrow H(\text{true})$ from Γ we abbreviate $X \vee (X^\frown\lceil S\rceil) \vee (X^\frown\lceil \neg S\rceil)$ to $next(X, S)$:

$$\left.\begin{array}{ll} & \vdots\\ k. & \Box\phi \Rightarrow H(\lceil\ \rceil)\end{array}\right\} \text{ deduction from } \Gamma$$

$$\left.\begin{array}{ll} & \vdots\\ l. & \Box\phi \Rightarrow (H(X) \Rightarrow H(next(X,S)))\end{array}\right\} \text{ deduction from } \Gamma$$

$$\begin{array}{lll} l+1. & \Box\phi \Rightarrow (H(X) \Rightarrow H(next(X,S)))\\ & \quad \Rightarrow ((\Box\phi \Rightarrow H(X)) \Rightarrow (\Box\phi \Rightarrow H(next(X,S)))) & \text{PL}\\ l+2. & (\Box\phi \Rightarrow H(X)) \Rightarrow (\Box\phi \Rightarrow H(next(X,S))) & l, l+1, \text{MP}\\ l+3. & \Box\phi \Rightarrow H(\text{true}) & k., l+2., \text{IR1}\end{array}$$

Note that it is taken into account that X does not occur in ϕ in the application of IR1 with $\Box\phi \Rightarrow H(X)$ as induction hypothesis.

Case IR2 is similar to IR1. □

The deduction theorem can often be used to simplify a proof. In connection with the application of the induction rules, the following theorem is convenient:

Theorem 3.3.

$$\Gamma, H(X) \vdash_{dc} H(X \vee (X^\frown\lceil S\rceil) \vee (X^\frown\lceil \neg S\rceil)) \text{ and } \Gamma \vdash_{dc} H(\lceil\ \rceil)$$
$$\text{implies } \Gamma \vdash_{dc} H(\text{true})$$

provided a deduction $\Gamma, H(X) \vdash_{dc} H(X \vee (X^\frown\lceil S\rceil) \vee (X^\frown\lceil \neg S\rceil))$ has the property that every application of the induction rules with hypothesis $H'(Y)$ satisfies that Y does not occur in $H(X)$.

Proof. Let $y_1, y_2, \ldots, y_n$ be all the variables occurring freely in $H(X)$ and let $H_c(X)$ denote the formula $(\forall y_1)(\forall y_2)\cdots(\forall y_n)H(X)$. Since $\Gamma \vdash_{dc} H(\lceil\rceil)$ and $\Gamma, H(X) \vdash_{dc} H(X \vee (X^\frown\lceil S\rceil) \vee (X^\frown\lceil\neg S\rceil))$ we also have $\Gamma \vdash_{dc} H_c(\lceil\rceil)$ and $\Gamma, H_c(X) \vdash_{dc} H_c(X \vee (X^\frown\lceil S\rceil) \vee (X^\frown\lceil\neg S\rceil))$ (using G and Q). In the following deduction we start using the deduction theorem:

	$\vdots$	
$k.$	$\Box H_c(X) \Rightarrow H_c(X \vee (X^\frown\lceil S\rceil) \vee (X^\frown\lceil\neg S\rceil))$	} deductions from Γ
	$\vdots$	
$l.$	$H_c(\lceil\rceil)$	
$l+1.$	$\Box H_c(\lceil\rceil)$	$l.$, R4
$l+2.$	$\Box H_c(X) \Rightarrow \Box H_c(X \vee (X^\frown\lceil S\rceil) \vee (X^\frown\lceil\neg S\rceil))$	$k.$, R6
$l+3.$	$\Box H_c(\text{true})$	$l+1., l+2.$, IR1
$l+4.$	$\Box H_c(\text{true}) \Rightarrow H_c(\text{true})$	T2
$l+5.$	$H_c(\text{true})$	$l+3., l+4.$, MP
$l+6$	$H(\text{true})$	$l+5.$, Q

where the application of IR1 uses $\Box H_c(X)$ as induction hypothesis. □

The following theorem is proven in a similar way:

Theorem 3.4.

$\Gamma, H(X) \vdash_{dc} H(X \vee (\lceil S\rceil^\frown X) \vee (\lceil\neg S\rceil^\frown X))$ and $\Gamma \vdash_{dc} H(\lceil\rceil)$ implies $\Gamma \vdash_{dc} H(\text{true})$

provided a deduction $\Gamma, H(X) \vdash_{dc} H(X \vee (\lceil S\rceil^\frown X) \vee (\lceil\neg S\rceil^\frown X))$ has the property that every application of the induction rules with hypothesis $H'(Y)$ satisfies that Y does not occur in $H(X)$.

The two induction rules can be used to prove:

(DC-T1) : $\lceil\rceil \vee (\text{true}^\frown\lceil S\rceil) \vee (\text{true}^\frown\lceil\neg S\rceil)$

(DC-T2) : $\lceil\rceil \vee (\lceil S\rceil^\frown\text{true}) \vee (\lceil\neg S\rceil^\frown\text{true})$

The proof of DC-T1 is easier using Theorem 3.3 with

$H(X) \mathrel{\widehat{=}} X \Rightarrow$ DC-T1

We establish:

$(X \Rightarrow \text{DC-T1}) \vdash_{dc} (X \vee (X^\frown\lceil S\rceil) \vee (X^\frown\lceil\neg S\rceil)) \Rightarrow \text{DC-T1}$

by establishing the three deductions:

a) $(X \Rightarrow \text{DC-T1}) \vdash_{dc} X \Rightarrow \text{DC-T1}$

b) $(X \Rightarrow \text{DC-T1}) \vdash_{dc} (X^\frown\lceil S\rceil) \Rightarrow \text{DC-T1}$

c) $(X \Rightarrow \text{DC-T1}) \vdash_{dc} (X^\frown\lceil\neg S\rceil) \Rightarrow \text{DC-T1}$

The first case, i.e. *a)*, is trivial. The cases *b)* and *c)* are similar, so we only establish one of them. The following constitutes a deduction for case *b)*:

1.	$X \Rightarrow \text{true}$	PL
2.	$(X^\frown\lceil S\rceil) \Rightarrow (\text{true}^\frown\lceil S\rceil)$	1., M1
3.	$(\text{true}^\frown\lceil S\rceil) \Rightarrow \text{DC-T1}$	PL
4.	$(X^\frown\lceil S\rceil) \Rightarrow \text{DC-T1}$	2., 3., PL

Having established *a*), *b*), and *c*) we have by PL:

$$(X \Rightarrow \text{DC-T1}) \vdash_{dc} (X \vee (X^\frown\lceil S\rceil) \vee (X^\frown\lceil\neg S\rceil)) \Rightarrow \text{DC-T1}, \text{ and}$$

$$\vdash_{dc} \lceil\,\rceil \Rightarrow \text{DC-T1}$$

Thus, we get true $\Rightarrow$ DC-T1 using Theorem 3.3, and then DC-T1 from PL.

4. Relative Completeness

In this section we consider the question whether there is a proof for every valid formula of DC, i.e. whether the proof system of DC is *complete*. Using DC formulas in specifications we want $\int S$ to be the integral of a real function. Therefore, to show completeness of DC, it must be shown that the axioms DC-A1 to DC-A6, together with the rules IR1 and IR2, and the axioms and rules of IL are enough to ensure that temporal variables of the form $\int S$ are definable by integrals.

In so doing, the functions and constants, e.g. + and 0, must be interpreted as real functions and constants, and the chop modality $^\frown$ occurring in the axioms must be interpreted as a modality, chopping intervals of real numbers.

Since every consistent formal system has a countable model, there is no way to axiomatize IL in order to ensure that functions, constants, etc. will be interpreted as real functions, constants, etc. Therefore, the best completeness result for DC that one can hope for is a *relative completeness* result, where valid IL formulas (wrt. a model based on real numbers) are taken as provable formulas. See also the remark on completeness in Section 2.3.

To formalize this notion, let $\mathcal{IL}$ be the set of all valid IL formulas, and define $\mathcal{IL}_{dc}$ to be the set of all DC instances of formulas of $\mathcal{IL}$, i.e. a formula $\varphi_{dc} \in \mathcal{IL}_{dc}$ is obtained from a formula $\varphi \in \mathcal{IL}$ as follows: Let $v_1, \ldots, v_n$ be the temporal variables occurring in φ, then φ_{dc} is obtained by replacing any occurrence of v_i with $\int S_i$, for some state expression S_i and for $1 \leqslant i \leqslant n$.

Each formula φ_{dc} is a valid DC formula since φ is a valid IL formula, and we will take $\mathcal{IL}_{dc}$ as the provable formula set of DC provided by IL.

Theorem 4.1. (Relative completeness) For any formula ϕ of DC:

$$\models_{dc} \phi \text{ implies } \mathcal{IL}_{dc} \vdash_{dc} \phi$$

We first sketch the main ideas of the proof of this theorem. The proof follows subsequently.

4.1. Proof Idea

For any valid DC formula ϕ, i.e. $\models_{dc} \phi$, we must show the existence of a deduction $\mathcal{IL}_{dc} \vdash_{dc} \phi$. Actually, we will give a deduction of $\mathcal{IL}_{dc} \vdash_{dcr} \phi$, where $\vdash_{dcr}$ denotes a deduction using the axioms of DC together with the two theorems DC-T1 and DC-T2, but not using the induction rules IR1 and IR2. We can, of course, construct a deduction $\mathcal{IL}_{dc} \vdash_{dc} \phi$ from a deduction $\mathcal{IL}_{dc} \vdash_{dcr} \phi$, since DC-T1 and DC-T2 are provable in DC.

A DC deduction $\mathcal{IL}_{dc} \vdash_{dcr} \phi$ can be considered to be an IL deduction

$$\mathcal{IL}_{dc} \cup DCR \vdash_{il} \phi$$

where DCR denotes the set of all instances of the axiom schemas DC-A1 to DC-A6 and instances of DC-T1 and DC-T2, where temporal variables have the form of durations. Thus, DCR denotes an infinite set of formulas.

However, for the given ϕ we construct an IL formula, say H_ϕ, having $v_1, v_2, \ldots$ as temporal variables with the property that a DC deduction $\mathcal{IL}_{dc} \vdash_{dcr} \phi$ can be constructed from an IL deduction $\mathcal{IL}, H_\phi \vdash_{il} \phi_h$, where ϕ_h is obtained from ϕ by "properly" replacing durations $\int S_i$ with temporal variables v_i. I.e. the formula H_ϕ provides an encoding in IL of the infinite set of formulas DCR.

Using the deduction theorem of IL we have that

$$\mathcal{IL}, H_\phi \vdash_{il} \phi_h \text{ iff } \mathcal{IL} \vdash_{il} \Box H_\phi \Rightarrow \phi_h$$

The main part of the proof is to show that $\Box H_\phi \Rightarrow \phi_h$ is a valid IL formula, i.e. an element of $\mathcal{IL}$, iff ϕ is a valid DC formula.

Because when $\models_{dc} \phi$, we have that $(\Box H_\phi \Rightarrow \phi_h) \in \mathcal{IL}$, and the DC formula $\Box H \Rightarrow \phi$, obtained from $\Box H_\phi \Rightarrow \phi_h$ by "properly" replacing temporal variables v_i with durations $\int S_i$, is a member of $\mathcal{IL}_{dc}$. Thus

$$\mathcal{IL}_{dc} \vdash_{dcr} \Box H \Rightarrow \phi$$

The formula H is a conjunction of DC axioms and instances of DC-T1 and DC-T2, and a deduction of $\mathcal{IL}_{dc} \vdash_{dcr} \phi$ is then easily achieved.

4.2. Proof of Relative Completeness

Let an arbitrary Duration Calculus formula ϕ be given. We now construct the IL formula H_ϕ.

Let $P_1, \ldots, P_l$ be the state variables occurring in ϕ, and let $\mathcal{S}$ be the set of state expressions which can be generated from these l state variables.

For $S \in \mathcal{S}$ let $[S] = \{S' \in \mathcal{S} \mid S \Leftrightarrow S' \textit{ in propositional logic }\}$, and let $\mathcal{S}_{\equiv}$ be the set of equivalence classes: $\{[S] \mid S \in \mathcal{S}\}$. The size k of $\mathcal{S}_{\equiv}$ is the number of Boolean functions in l variables, i.e. $k = 2^{2^l}$.

Select k temporal variables $v_1, \ldots, v_k$ and put them in one-to-one correspondence with the equivalence classes. We can thus index the selected temporal variables with equivalence classes. Furthermore, we assume a representative for each equivalence class to be given in the following. (This could for instance be given by a disjunctive normal form of state expressions.)

For the axiom schemas (DC-A1) to (DC-A5), and for the two theorem schemas (DC-T1) and (DC-T2) we construct seven finite sets of IL formulas:

$$\begin{aligned}
\mathcal{H}_1 &\mathrel{\hat{=}} \{v_{[0]} = 0\} \\
\mathcal{H}_2 &\mathrel{\hat{=}} \{v_{[1]} = \ell\} \\
\mathcal{H}_3 &\mathrel{\hat{=}} \{v_{[S]} \geqslant 0 \mid [S] \in \mathcal{S}_{\equiv}\} \\
\mathcal{H}_4 &\mathrel{\hat{=}} \{v_{[S_1]} + v_{[S_2]} = v_{[S_1 \vee S_2]} + v_{[S_1 \wedge S_2]} \mid [S_1], [S_2] \in \mathcal{S}_{\equiv}\} \\
\mathcal{H}_5 &\mathrel{\hat{=}} \{(\forall x)(\forall y)(((v_{[S]} = x)^\frown(v_{[S]} = y)) \Rightarrow (v_{[S]} = x + y)) \mid [S] \in \mathcal{S}_{\equiv}\} \\
\mathcal{H}_6 &\mathrel{\hat{=}} \{\lceil\!\rceil \vee (\text{true}^\frown\lceil v_{[S]}\rceil) \vee (\text{true}^\frown\lceil v_{[\neg S]}\rceil) \mid [S] \in \mathcal{S}_{\equiv}\} \\
\mathcal{H}_7 &\mathrel{\hat{=}} \{\lceil\!\rceil \vee (\lceil v_{[S]}\rceil^\frown\text{true}) \vee (\lceil v_{[\neg S]}\rceil^\frown\text{true}) \mid [S] \in \mathcal{S}_{\equiv}\}
\end{aligned}$$

where we define $\lceil v_{[S]}\rceil$ by $v_{[S]} = v_{[1]} \wedge v_{[1]} > 0$.

Define H_ϕ to be the conjunction of all the formulas in $\mathcal{H}_1$ to $\mathcal{H}_7$, and let ϕ_h be the IL formula obtained from ϕ by replacing each duration $\int S$ by $v_{[S]}$.

The definition and lemmas below are convenient for the completeness proof.

Definition. We call a triple $(\mathcal{J}, \mathcal{V}, [b,e])$ for an *H-triple* if

$$\mathcal{J}, \mathcal{V}, [b,e] \models \Box H_\phi$$

i.e. if for any sub-interval $[c,d]$ of $[b,e]$: $\mathcal{J}, \mathcal{V}, [c,d] \models H_\phi$.

Notation: When an interpretation $\mathcal{J}$ to temporal variables is given in the context, we do not mention $\mathcal{J}$ explicitly and write $\underline{v}$ for $\mathcal{J}(v)$.

Lemma 4.1. Given an H-triple $(\mathcal{J}, \mathcal{V}, [b,e])$. Then

(*i*) $0 \leqslant \underline{v}_{[S]}[c,d] \leqslant d-c$
(*ii*) $\underline{v}_{[S]}[c,d] = (d-c) - \underline{v}_{[\neg S]}[c,d]$
(*iii*) $\underline{v}_{[S_1]}[c,d] \leqslant \underline{v}_{[S_1 \vee S_2]}[c,d]$

for any $S, S_1, S_2 \in \mathcal{S}$ and any sub-interval $[c,d]$ of $[b,e]$.

Proof. (*i*) and (*ii*) are trivial. Since $\neg S_1 \vee (S_1 \vee S_2)$ is a tautology, we have from $\mathcal{H}_2$ that

$$\underline{v}_{[1]}[c,d] = (d-c) = \underline{v}_{[\neg S_1 \vee (S_1 \vee S_2)]}[c,d]$$

From $\mathcal{H}_4$ we have

$$\underline{v}_{[\neg S_1]}[c,d] + \underline{v}_{[(S_1 \vee S_2)]}[c,d] = \underline{v}_{[\neg S_1 \vee (S_1 \vee S_2)]}[c,d] + \underline{v}_{[\neg S_1 \wedge (S_1 \vee S_2)]}[c,d]$$

i.e. using (*ii*) we get

$$(d-c) - \underline{v}_{[S_1]}[c,d] + \underline{v}_{[(S_1 \vee S_2)]}[c,d] = (d-c) + \underline{v}_{[\neg S_1 \wedge (S_1 \vee S_2)]}[c,d]$$

which gives $\underline{v}_{[S_1]}[c,d] \leqslant \underline{v}_{[S_1 \vee S_2]}[c,d]$ (since $\underline{v}_{[\neg S_1 \wedge (S_1 \vee S_2)]}[c,d] \geqslant 0$ by $\mathcal{H}_3$). □

Definition. Let $S \in \mathcal{S}$ and $(\mathcal{J}, \mathcal{V}, [b,e])$, where $b < e$, be an H-triple. A *partition of* $[b,e]$ *for* S is a finite partition $b = t_0 < t_1 < \ldots < t_n = e$ of $[b,e]$ so that for $i = 1, \ldots n$ we get:

$$\text{either } \mathcal{J}, \mathcal{V}, [t_{i-1}, t_i] \models \lceil v_{[S]} \rceil \text{ or } \mathcal{J}, \mathcal{V}, [t_{i-1}, t_i] \models \lceil v_{[\neg S]} \rceil$$

Lemma 4.2. Let $(\mathcal{J}, \mathcal{V}, [b,e])$, where $b < e$, be an H-triple. For any $S \in \mathcal{S}$, there is a finite partition of $[b,e]$ for S.

Proof. For any $t : b < t < e$, there are (by $\mathcal{H}_6$ and $\mathcal{H}_7$) t' and t'' so that $b \leqslant t' < t < t'' \leqslant e$ and

$$\left.\begin{array}{l} \mathcal{J}, \mathcal{V}, [t', t] \models \lceil v_{[S]} \rceil \text{ or } \mathcal{J}, \mathcal{V}, [t', t] \models \lceil v_{[\neg S]} \rceil \\ \qquad\qquad \text{and} \\ \mathcal{J}, \mathcal{V}, [t, t''] \models \lceil v_{[S]} \rceil \text{ or } \mathcal{J}, \mathcal{V}, [t, t''] \models \lceil v_{[\neg S]} \rceil \end{array}\right\} \qquad (\dagger)$$

Thus, there is an open interval (t', t'') covering t (but not b nor e) so that the closed interval $[t', t'']$ has the above property (†).

For the left end point b, there is by $\mathcal{H}_7$ a t'' so that $b < t'' \leqslant e$ and

$$\mathcal{J}, \mathcal{V}, [b, t''] \models \lceil v_{[S]} \rceil \text{ or } \mathcal{J}, \mathcal{V}, [b, t''] \models \lceil v_{[\neg S]} \rceil \qquad (\dagger_b)$$

Thus, there is an open interval (t', t'') covering b so that the closed interval $[b, t'']$ has the above property $(\dagger_b)$. (Select arbitrary $t' < b$.)

Similarly for e, there is by $\mathcal{H}_6$ a t' so that $b \leqslant t' < e$ and

$$\mathcal{J}, \mathcal{V}, [t', e] \models \lceil v_{[S]} \rceil \text{ or } \mathcal{J}, \mathcal{V}, [t', e] \models \lceil v_{[\neg S]} \rceil \qquad (\dagger_e)$$

Thus, there is an open interval (t', t'') covering e so that the closed interval $[t', e]$ has the above property $(\dagger_e)$. (Select arbitrary $t'' > e$.)

So we have an infinite collection of open intervals covering the closed and bounded interval $[b, e]$. Then by Heine-Borels theorem there is a finite sub-collection $C = \{I_1, \ldots I_m\}$ of the open intervals covering $[b, e]$.

STEP 1: Select the open interval $I_i = (a_i, b_i)$ from C covering b. Then the closed interval $[b, b_i]$ satisfies by $(\dagger_b)$:

$$\mathcal{J}, \mathcal{V}, [b, b_i] \models \lceil v_{[S]} \rceil \text{ or } \mathcal{J}, \mathcal{V}, [b, b_i] \models \lceil v_{[\neg S]} \rceil.$$

STEP 2: Stop if $b_i = e$. Otherwise $b_i < e$. Select an open interval $I_j = (a_j, b_j)$ from C covering b_i. If $e < b_j$, then by $(\dagger_e)$ either

$$\mathcal{J}, \mathcal{V}, [b_i, e] \models \lceil v_{[S]} \rceil \text{ or } \mathcal{J}, \mathcal{V}, [b_i, e] \models \lceil v_{[\neg S]} \rceil$$

will hold for the closed interval $[b_i, e]$ and we stop.

If $b_j \leqslant e$, then the closed interval $[b_i, b_j]$ will by $(\dagger)$ satisfy one of

1 : $\mathcal{J}, \mathcal{V}, [b_i, b_j] \models \lceil v_{[S]} \rceil$

2 : $\mathcal{J}, \mathcal{V}, [b_i, b_j] \models \lceil v_{[\neg S]} \rceil$

3 : $\mathcal{J}, \mathcal{V}, [b_i, m] \models \lceil v_{[S]} \rceil$ and $\mathcal{J}, \mathcal{V}, [m, b_j] \models \lceil v_{[\neg S]} \rceil$, for some $m : b_i < m < b_j$

4 : $\mathcal{J}, \mathcal{V}, [b_i, m] \models \lceil v_{[\neg S]} \rceil$ and $\mathcal{J}, \mathcal{V}, [m, b_j] \models \lceil v_{[S]} \rceil$, for some $m : b_i < m < b_j$

Repeat *STEP 2* until a partition of $[b, e]$ is achieved. This terminates since there is a finite number of open intervals in C. □

Lemma 4.3. An H-triple $(\mathcal{J}, \mathcal{V}, [b, e])$ where $b < e$ induces an interpretation $\mathcal{I}$, so that for any $S \in \mathcal{S}$ and $t \in [b, e)$:

$$S_{\mathcal{I}}(t) = \begin{cases} 1, & \text{if } t \in [t_{i-1}, t_i) \text{ and } \mathcal{J}, \mathcal{V}, [t_{i-1}, t_i] \models \lceil v_{[S]} \rceil \\ 0, & \text{if } t \in [t_{i-1}, t_i) \text{ and } \mathcal{J}, \mathcal{V}, [t_{i-1}, t_i] \models \lceil v_{[\neg S]} \rceil \end{cases}$$

for some partition $b = t_0 < t_1 < \ldots < t_n = e$ of $[b, e]$ for S.

Proof. Define an interpretation $\mathcal{I}$ as follows: Let $Q_{\mathcal{I}}(t) = 0, t \in \mathbb{T}\text{ime}$, for any state variable $Q \notin \mathcal{S}$. For any state variable $P \in \mathcal{S}$, let $b = t_0 < t_1 < \ldots < t_n = e$ be a partition of $[b, e]$ for P given by Lemma 4.2. Let

$$P_{\mathcal{I}}(t) = \begin{cases} 1, & \text{if } t_{i-1} \leqslant t < t_i, \mathcal{J}, \mathcal{V}, [t_{i-1}, t_i] \models \lceil v_{[P]} \rceil, \text{ and } 1 \leqslant i \leqslant n \\ 0, & \text{otherwise} \end{cases}$$

Each such function has only a finite number of discontinuity points in any interval, so $\mathcal{I}$ is indeed an interpretation.

We prove the remaining parts of the lemma by structural induction on S. If $S \notin \mathcal{S}$ there is nothing to prove, so assume below that $S \in \mathcal{S}$. The cases where S is 0, 1, or P are trivial, so consider:

CASE: S has the form $\neg S'$.

Let $b = t_0 < t_1 < \ldots < t_n = e$ be a partition of $[b, e]$ for S' given by the induction hypothesis. This is also a partition for $\neg S'$, as $\neg\neg S' \Leftrightarrow S'$.

Consider an arbitrary $t, b \leqslant t < e$. I.e. $t_{i-1} \leqslant t < t_i$ for some $i \in \{1, \ldots, n\}$. By definition we have that $(\neg S')_{\mathcal{I}}(t) = 1 - S'_{\mathcal{I}}(t)$.

If $\mathcal{J}, \mathcal{V}, [t_{i-1}, t_i] \models \lceil v_{[\neg\neg S']} \rceil$, then $(\neg S')_{\mathcal{I}}(t) = 0$ as we have $S'_{\mathcal{I}}(t) = 1$ from the induction hypothesis.

If $\mathcal{J}, \mathcal{V}, [t_{i-1}, t_i] \models \lceil v_{[\neg S']} \rceil$, then $S'_{\mathcal{I}}(t) = 0$ (induction hypothesis). But then $(\neg S')_{\mathcal{I}}(t) = 1$ as required.

CASE: S has the form $S' \vee S''$.

We combine two partitions of $[b, e]$ for S and S' given by the induction hypothesis to get a finite partition $b = t_0 < t_1 < \ldots < t_n = e$, where precisely one of the four formulas: $\lceil v_{[S']} \rceil \wedge \lceil v_{[S'']} \rceil$, $\lceil v_{[\neg S']} \rceil \wedge \lceil v_{[\neg S'']} \rceil$, $\lceil v_{[S']} \rceil \wedge \lceil v_{[\neg S'']} \rceil$, or $\lceil v_{[\neg S']} \rceil \wedge \lceil v_{[S'']} \rceil$, will hold in each section. Furthermore, each section $[t_{i-1}, t_i]$ will match one of the cases:

(i) $\underline{v}_{[S']} = \underline{v}_{[S'']} = t_i - t_{i-1}$ and $S'_{\mathcal{I}}(t) = S''_{\mathcal{I}}(t) = 1$, i.e. $(S' \vee S'')_{\mathcal{I}}(t) = 1$, for $t_{i-1} \leqslant t < t_i$.

(ii) $\underline{v}_{[\neg S']} = \underline{v}_{[\neg S'']} = t_i - t_{i-1}$, $S'_{\mathcal{I}}(t) = S''_{\mathcal{I}}(t) = 0$, i.e. $(S' \vee S'')_{\mathcal{I}}(t) = 0$, for $t_{i-1} \leqslant t < t_i$.

(iii) $\underline{v}_{[S']} = t_i - t_{i-1}$, $\underline{v}_{[\neg S'']} = t_i - t_{i-1}$, $S'_{\mathcal{I}}(t) = 1$ and $S''_{\mathcal{I}}(t) = 0$, i.e. $(S' \vee S'')_{\mathcal{I}}(t) = 1$, for $t_{i-1} \leqslant t < t_i$.

(iv) $\underline{v}_{[\neg S']} = t_i - t_{i-1}$, $\underline{v}_{[S'']} = t_i - t_{i-1}$, $S'_{\mathcal{I}}(t) = 0$ and $S''_{\mathcal{I}}(t) = 1$, i.e. $(S' \vee S'')_{\mathcal{I}}(t) = 1$, for $t_{i-1} \leqslant t < t_i$.

We prove that $b = t_0 < t_1 < \ldots < t_n = e$ is a partition of $[b, e]$ for $S' \vee S''$ by considering the four cases:

(i) We must prove that: $\mathcal{J}, \mathcal{V}, [t_{i-1}, t_i] \models \lceil v_{[S' \vee S'']} \rceil$. From Lemma 4.1:

$$0 \leqslant \underline{v}_{[S' \vee S'']}[t_{i-1}, t_i] \leqslant t_i - t_{i-1} \quad \text{and} \quad 0 \leqslant v_{[S' \wedge S'']}[t_{i-1}, t_i] \leqslant t_i - t_{i-1}$$

so it follows from $\mathcal{H}_4$ that $\underline{v}_{[S' \vee S'']}[t_{i-1}, t_i] = \underline{v}_{[S' \wedge S'']}[t_{i-1}, t_i] = t_i - t_{i-1}$. Using the definition of $\lceil v_{[S' \vee S'']} \rceil$ we have that $\mathcal{J}, \mathcal{V}, [t_{i-1}, t_i] \models \lceil v_{[S' \vee S'']} \rceil$.

(ii) We must prove that: $\mathcal{J}, \mathcal{V}, [t_{i-1}, t_i] \models \lceil v_{[\neg(S' \vee S'')]} \rceil$. Since (from $\mathcal{H}_3$)

$$\underline{v}_{[S' \vee S'']}[t_{i-1}, t_i] \geqslant 0 \quad \text{and} \quad \underline{v}_{[S' \wedge S'']}[t_{i-1}, t_i] \geqslant 0$$

it follows from $\mathcal{H}_4$ and Lemma 4.1 that $\underline{v}_{[S' \vee S'']}[t_{i-1}, t_i] = 0$. Therefore, by Lemma 4.1 $\underline{v}_{[\neg(S' \vee S'')]}[t_{i-1}, t_i] = t_i - t_{i-1}$ and hence $\mathcal{J}, \mathcal{V}, [t_{i-1}, t_i] \models \lceil v_{[\neg(S' \vee S'')]} \rceil$.

(iii) We must prove that: $\mathcal{J}, \mathcal{V}, [t_{i-1}, t_i] \models \lceil v_{[S' \vee S'']} \rceil$. Since by Lemma 4.1

$$\underline{v}_{[S']}[t_{i-1}, t_i] \leqslant \underline{v}_{[S' \vee S'']}[t_{i-1}, t_i] \leqslant t_i - t_{i-1}$$

it follows that $\underline{v}_{[S' \vee S'']}[t_{i-1}, t_i] = t_i - t_{i-1}$. I.e. $\mathcal{J}, \mathcal{V}, [t_{i-1}, t_i] \models \lceil v_{[S' \vee S'']} \rceil$.

(iv) Similar to (iii).

□

Theorem 4.2. For a given H-triple $(\mathcal{J}, \mathcal{V}, [b, e])$, there is an interpretation $\mathcal{I}$ so that for any $S \in \mathcal{S}$ and interval $[c, d] \subseteq [b, e]$:

$$\mathcal{I}[\![\int S]\!][c, d] = \underline{v}_{[S]}[c, d]$$

Proof. If $b = e$, then any interpretation $\mathcal{I}$ will do, since $c = d$, $\mathcal{I}[\![\int S]\!][c, d] = 0$ and $\underline{v}_{[S]}[c, d] = 0$, because $0 \leqslant \underline{v}_{[S]}[c, d] \leqslant d - c$ from Lemma 4.1. So suppose that $b < e$. Let $\mathcal{I}$ be an interpretation given by Lemma 4.3. The case where $c = d$

is treated as above, so suppose that $c < d$. Let $c = t_0 < t_1 < \ldots < t_n = d$ be a partition of $[c,d]$ for S. We have that for any $t \in [c,d)$:

$$S_{\mathcal{I}}(t) = \begin{cases} 1, & \text{if } t \in [t_{i-1}, t_i) \text{ and } \mathcal{J},\mathcal{V},[t_{i-1},t_i] \models [\![v_{[S]}]\!] \\ 0, & \text{if } t \in [t_{i-1}, t_i) \text{ and } \mathcal{J},\mathcal{V},[t_{i-1},t_i] \models [\![v_{[\neg S]}]\!] \end{cases}$$

Thus $\int_{t_{i-1}}^{t_i} S_{\mathcal{I}}(t)dt = \underline{v}_{[S]}[t_{i-1}, t_i]$, for $i = 1,\ldots,n$, and by $\mathcal{H}_5$:

$$\mathcal{I}[\![\int S]\!][c,d] = \sum_{i=1}^{n} \underline{v}_{[S]}[t_{i-1}, t_i] = \underline{v}_{[S]}[c,d]$$

□

Let ϕ_h be the IL formula obtained from ϕ by replacing any occurrence of $\int S_i$ in ϕ with $v_{[S_i]}$.

Theorem 4.3.

$$\models_{dc} \phi \text{ iff } \models (\Box H_\phi) \Rightarrow \phi_h$$

Proof. We first prove that $\models_{dc} \phi$ implies $\models (\Box H_\phi) \Rightarrow \phi_h$, so suppose that $\not\models (\Box H_\phi) \Rightarrow \phi_h$, i.e. there is an H-triple $(\mathcal{J},\mathcal{V},[b,e])$ so that $\mathcal{J},\mathcal{V},[b,e] \not\models \phi_h$. By Theorem 4.2 there is an interpretation $\mathcal{I}$ so that for any $S \in \mathcal{S}$ and interval $[c,d] \subseteq [b,e]$:

$$\mathcal{I}[\![\int S]\!][c,d] = \underline{v}_{[S]}[c,d]$$

Since $\mathcal{J},\mathcal{V},[b,e] \not\models \phi_h$, we have that $\mathcal{I},\mathcal{V},[b,e] \not\models_{dc} \phi$, and hence $\not\models_{dc} \phi$.

To prove the other direction, i.e. $\models (\Box H_\phi) \Rightarrow \phi_h$ implies $\models_{dc} \phi$, suppose that $\not\models_{dc} \phi$, i.e. there is an interpretation $\mathcal{I}$, value assignment $\mathcal{V}$ and interval $[b,e]$ so that $\mathcal{I},\mathcal{V},[b,e] \not\models_{dc} \phi$. Let us construct an interpretation $\mathcal{J}$ so that $\underline{v}_{[S]}[c,d] = \mathcal{I}[\![\int S]\!][c,d]$ for all $S \in \mathcal{S}$ and intervals $[c,d]$. (By axiom (DC-A6) this is well-defined.) By construction we have that $\mathcal{J},\mathcal{V},[b,e] \not\models \phi_h$ and from Theorem 3.1 (soundness) $\mathcal{J},\mathcal{V},[b,e] \models \Box H_\phi$. So $\not\models (\Box H_\phi) \Rightarrow \phi_h$. □

The relative completeness theorem can now be proven:

Theorem 4.4. (Relative completeness) For any formula ϕ of DC

$$\models_{dc} \phi \text{ implies } \mathcal{IL}_{dc} \vdash_{dc} \phi$$

Proof. Suppose $\models_{dc} \phi$. By Theorem 4.3 we get $\models (\Box H_\phi) \Rightarrow \phi_h$. Let H be obtained from H_ϕ by replacing each $v_{[S]}$ by $\int S$. Then $(\Box H \Rightarrow \phi) \in \mathcal{IL}_{dc}$ and

$$\mathcal{IL}_{dc} \vdash_{dc} \Box H \Rightarrow \phi$$

We have that H is a conjunction of DC axioms and instances of DC-T1 and DC-T2 and therefore by PL and R4:

$$\vdash_{dc} \Box H$$

and a deduction of $\mathcal{IL}_{dc} \vdash_{dc} \phi$ follows by applying MP. □

Remark: Note that the relative completeness result was achieved using the theorems DC-T1 and DC-T2 instead of the two induction rules IR1 and IR2. It is, however, convenient to have the two induction when conducting proofs. □

5. Decidability

In this section we consider subsets of formulas of DC for which it is decidable whether a formula from the subset is satisfiable. Since a formula ϕ is valid iff the formula $\neg\phi$ is not satisfiable, we can decide whether a formula in the subset is valid as well. The decidability results presented are based on [ZHS93].

It turns out, as shall be shown in the next section, that even very simple subsets of formulas of DC are undecidable.

We investigate now the set *RDC* of formulas generated by

1. if S is a state expression, then $\lceil S \rceil \in RDC$
2. if $\phi, \psi \in RDC$, then $\neg\phi, \phi \vee \psi, \phi^\frown\psi \in RDC$

We present a *discrete time* version of DC together with decidability results for satisfiability of *RDC* formulas, since the denseness of the time domain turns out to be a source of complication. It is shown that *RDC* is expressive enough to formalize an interesting case study under a discrete time interpretation.

This section is organized into five subsections, where the first presents a case study, the second develops a discrete time version of DC, and the third gives a decidability result for *RDC* with regard to discrete time. A decidable result for *RDC* with regard to continuous time is presented in subsection four, and the last subsection discusses the complexity of the decision algorithms.

5.1. A Case Study

In this section we formalize a set of requirements for a gas burner system. These requirements, which are presented in [RRH93], are later shown to be expressible in *RDC* in discrete time.

The state variables used in the requirements are:

$$\mathrm{G,F,H,I} \in \mathbb{T}\mathrm{ime} \to \{0,1\}$$

where G and F describe gas and flame as in Section 1. The intention of the state variable H is that it is 1 iff there is a heat request to the gas burner and the intention with the state variable I is that it is 1 iff an ignition transformer is on, i.e. it is trying to ignite the gas.

The requirements are:

1. For intervals less than 30*s*, the gas may leak for at most 4*s*:

   $$Req_1 \mathrel{\widehat{=}} \ell \leqslant 30 \Rightarrow \textstyle\int \mathrm{L} \leqslant 4$$

   where $\mathrm{L} \mathrel{\widehat{=}} \mathrm{G} \wedge \neg\mathrm{F}$.
2. Heat request off must in 60*s* result in the flame being off:

   $$Req_2 \mathrel{\widehat{=}} \lceil\neg\mathrm{H}\rceil \Rightarrow (\ell \leqslant 60 \vee ((\ell \leqslant 60)^\frown\lceil\neg\mathrm{F}\rceil))$$

3. Heat request on must in 60*s* result in the flame burning, unless either an ignition failure or a flame failure has occurred:

   $$Req_3 \mathrel{\widehat{=}}$$
   $$\lceil\mathrm{H}\rceil \Rightarrow (\ell \leqslant 60 \vee ((\ell \leqslant 60)^\frown\lceil\mathrm{F}\rceil^\frown\mathrm{true}) \vee \Diamond\neg \mathit{FlOK} \vee \Diamond\neg \mathit{IgOK})$$

   where the flame is working correctly if it does not disappear while the gas is supplied:

$$FlOK \mathrel{\widehat{=}} \lceil G \rceil \Rightarrow \neg\Diamond(\lceil F \rceil^\frown \lceil \neg F \rceil)$$

and the ignition is working correctly if the gas ignites in $1s$:

$$IgOK \mathrel{\widehat{=}} \lceil G \wedge I \rceil \Rightarrow (\ell \leqslant 1 \vee ((\ell \leqslant 1)^\frown \lceil F \rceil))$$

The requirements which must be met by the gas burner system is the conjunction of the above requirements, i.e.

$$Req \mathrel{\widehat{=}} Req_1 \wedge Req_2 \wedge Req_3$$

This ends our presentation of the case study. In [RRH93] it is developed further over several steps, ending with a specification of an implementation of the above requirements. The implementation consists of a control program interacting with sensors measuring heat request and flame, and actuators governing the gas and the ignition devices. The implementation is specified as a DC formula, and this specification is shown to imply the above requirements.

5.2. Discrete Time Duration Calculus

What shall we consider to be a discrete time Duration Calculus?

Even when the set of natural numbers $\mathbb{N} = \{0, 1, 2, \ldots\}$ is chosen as the discrete structure, questions remain concerning restrictions on interpretations, intervals, and the truth of formulas.

First of all, it is required for any interpretation:

$$\mathcal{I} \in SVar \to (\mathbb{T}\text{ime} \to \{0, 1\})$$

that the set of discontinuity points of each $P_{\mathcal{I}}$ must be a subset of $\mathbb{N}$. An interpretation satisfying this property is called a *discrete interpretation.* Note that we omit predicate letters from interpretations, since there are no predicate letters in formulas of *RDC*.

Likewise, we will only consider *discrete intervals* $[b, e] \subset \mathbb{I}\text{ntv}$ where $b, e \in \mathbb{N}$.

Finally, for a given DC formula ϕ, its truth value is only considered in discrete intervals of discrete interpretations.

As a consequence of this, the definition of chop $(\phi^\frown\psi)$ is different from the one given in Section 2.2 for continuous time. Assume that $\mathcal{I}$ is a discrete interpretation and $[b, e]$ is a discrete interval:

$$\mathcal{I}, [b, e] \models_{dc} \phi^\frown\psi \quad \text{iff} \quad \mathcal{I}, [b, m] \models_{dc} \phi \text{ and } \mathcal{I}, [m, e] \models_{dc} \psi, \text{ for some } m \in [b, e] \text{ where } m \in \mathbb{N}$$

where we leave out value assignments $(\mathcal{V})$ from the definition since we have no global variables in formulas of *RDC*.

The other semantic clauses are not given, as they remain as in Section 2.2.

A DC formula ϕ is *valid for discrete time* iff $\mathcal{I}, [b, e] \models_{dc} \phi$ for any discrete interpretation $\mathcal{I}$ and any discrete interval $[b, e]$, and ϕ is *satisfiable for discrete time* iff $\mathcal{I}, [b, e] \models_{dc} \phi$, for some discrete interpretation $\mathcal{I}$ and some discrete interval $[b, e]$.

5.2.1. Discrete Time versus Continuous Time

One can ask the question what difference it makes to consider a discrete time domain instead of a continuous time domain?

The following DC formula is valid for discrete time, but it is not valid for the continuous time semantics:

$$\ell = 1 \Leftrightarrow (\ell > 0 \wedge \neg((\ell > 0)^\frown(\ell > 0)))$$

There are also formulas of DC which are valid for continuous time, but not valid for discrete time, e.g.

$$\ell > 0 \Rightarrow ((\ell > 0)^\frown(\ell > 0))$$

Thus, there is no simple relationship between the formulas which are valid for discrete time and the formulas which are valid for continuous time.

5.2.2. Expressibility of the case in RDC

Due to the following equivalences

$$\begin{aligned}
\ell = 0 &\Leftrightarrow \neg\lceil 1\rceil \\
\ell = 1 &\Leftrightarrow \lceil 1\rceil \wedge \neg(\lceil 1\rceil^\frown\lceil 1\rceil) \\
\text{true} &\Leftrightarrow \ell = 0 \vee \neg(\ell = 0) \\
\int P = 0 &\Leftrightarrow \lceil\neg P\rceil \vee \ell = 0 \\
\int P = 1 &\Leftrightarrow (\int P = 0)^\frown(\lceil P\rceil \wedge \ell = 1)^\frown(\int P = 0) \\
\int P = k+1 &\Leftrightarrow (\int P = k)^\frown(\int P = 1) \quad (k \geqslant 1, k \in \mathbb{N}) \\
\int P \geqslant k &\Leftrightarrow (\int P = k)^\frown\text{true} \quad (k \in \mathbb{N}) \\
\int P > k &\Leftrightarrow (\int P \geqslant k) \wedge \neg(\int P = k) \\
\int P \leqslant k &\Leftrightarrow \neg(\int P > k) \\
\int P < k &\Leftrightarrow (\int P \leqslant k) \wedge \neg(\int P = k)
\end{aligned}$$

and since $\ell = \int 1$, we can encode the formula *Req* in *RDC*.

5.3. Decidability for Discrete Time

We show that satisfiability of a formula $\phi \in RDC$ is decidable by defining a regular language $\mathcal{L}_1(\phi)$, so that

ϕ is satisfiable iff $\mathcal{L}_1(\phi)$ is non-empty.

Let $\mathcal{S}$ be the (finite) set of state variables occurring in ϕ. Then the *alphabet* Σ of the language $\mathcal{L}_1(\phi)$ is the set $\Sigma = \mathcal{P}(\mathcal{S})$ of subsets of $\mathcal{S}$. A letter $a \in \Sigma$ is called a *basic conjunct* and is interpreted as the state expression:

$$\bigwedge\{P \mid P \in a\} \wedge \bigwedge\{\neg Q \mid Q \in (\mathcal{S} \setminus a)\}$$

which asserts that all state variables in a are one, and those not in a are zero.

The disjunctive normal form of a state expression S is a disjunction $\bigvee_{i=1}^{n} a_i$ of basic conjuncts, $n \geqslant 0$. We let $DNF(S) = \{a_1, \ldots, a_n\} \subseteq \Sigma$ denote the uniquely determined set of basic conjuncts of the disjunctive normal form of S.

We represent a formula ϕ by a regular language $\mathcal{L}_1(\phi) \subseteq \Sigma^*$, so that ϕ holds on a discrete interval $[b, e]$ for a given discrete interpretation $\mathcal{I}$ iff there is a string

$v \in \mathcal{L}_1(\phi)$ which corresponds to the interpretation $\mathcal{I}$ on $[b,e]$. A letter $a \in \Sigma$ describes a unit interval. The formula ϕ is satisfiable for discrete time iff the language $\mathcal{L}_1(\phi)$ is non-empty. Since emptiness of a regular language is decidable, we obtain a procedure for deciding satisfiability of ϕ.

The definition of $\mathcal{L}_1(\phi)$ is quite straightforward. An "almost everywhere" formula $\lceil S \rceil$ is represented by the positive closure B^+, where $B = DNF(S)$ is the set of basic conjuncts of S. Disjunction $\phi \vee \psi$ is represented by union, negation $\neg\phi$ by complement, and chop $\phi^\frown\psi$ by concatenation. Since B^+ is a regular language, and the family of regular languages is closed under union, complement, and concatenation, every formula is represented by a regular language. More precisely,

$$\begin{aligned}
\mathcal{L}_1(\lceil S \rceil) &= (DNF(S))^+ \\
\mathcal{L}_1(\phi \vee \psi) &= \mathcal{L}_1(\phi) \cup \mathcal{L}_1(\psi) \\
\mathcal{L}_1(\neg\phi) &= \Sigma^* \setminus \mathcal{L}_1(\phi) \\
\mathcal{L}_1(\phi^\frown\psi) &= \mathcal{L}_1(\phi)\mathcal{L}_1(\psi)
\end{aligned}$$

The last line uses the concatenation $L_1L_2 = \{vu | v \in L_1, u \in L_2\}$ of two regular languages L_1 and L_2. Positive closure, union, complement, and concatenation of regular languages can be realized by operations on finite state automata [HoU79].

String $v = a_1 \ldots a_N \in \Sigma^*$ *corresponds* to a discrete interpretation $\mathcal{I}$ on a discrete interval $[0,N]$ if $\mathcal{I}[\![a_i]\!](t) = 1$ for $t \in (i-1,i)$, $i \in \{1,\ldots,N\}$. (If $N = 0$ then $v = \epsilon$ is the empty string).

Lemma 5.1. Let a formula $\phi \in RDC$, a discrete interpretation $\mathcal{I}$, and a corresponding string $v = a_1 \ldots a_N$ be given. Then $\mathcal{I}, [0,N] \models_{dc} \phi$ for discrete time iff v belongs to $\mathcal{L}_1(\phi)$.

Proof. By induction on the structure of ϕ. □

Now for any string v in Σ^* there is an interpretation $\mathcal{I}$ so that v corresponds to $\mathcal{I}$, and conversely: for any interpretation $\mathcal{I}$ and interval $[0,N]$ there is a string $v = a_1 \ldots a_N$ in Σ^* which corresponds to $\mathcal{I}$. Thus by 5.1, we have

Lemma 5.2. A formula $\phi \in RDC$ is satisfiable for discrete time iff the regular language $\mathcal{L}_1(\phi)$ is non-empty.

Theorem 5.1. The satisfiability question in discrete time of DC formulas in *RDC* is decidable.

We show now how to prove or disprove the validity of formulas.

Question 1: Is the formula $(\lceil P \rceil^\frown\lceil P \rceil) \Rightarrow \lceil P \rceil$ valid for discrete time?

Since P is the only state variable occurring in the formula, the alphabet $\Sigma = \{\{P\},\{\}\}$. We have

$$\begin{aligned}
&(\lceil P \rceil^\frown\lceil P \rceil) \Rightarrow \lceil P \rceil \text{ is valid} \\
\text{iff}\quad &\neg((\lceil P \rceil^\frown\lceil P \rceil) \Rightarrow \lceil P \rceil) \text{ is not satisfiable} \\
\text{iff}\quad &(\lceil P \rceil^\frown\lceil P \rceil) \wedge \neg\lceil P \rceil \text{ is not satisfiable} \\
\text{iff}\quad &\mathcal{L}_1(\lceil P \rceil^\frown\lceil P \rceil) \cap \mathcal{L}_1(\neg\lceil P \rceil) = \{\} \\
\text{iff}\quad &\{\{P\}^i \mid i \geqslant 2\} \cap (\Sigma^* \setminus \{\{P\}^i \mid i \geqslant 1\}) = \{\}
\end{aligned}$$

which holds. Therefore $(\lceil P \rceil^\frown\lceil P \rceil) \Rightarrow \lceil P \rceil$ is valid for discrete time. □

Question 2: Is the formula $\lceil P \rceil \Rightarrow (\lceil P \rceil^\frown \lceil P \rceil)$ valid for discrete time?

Again, the alphabet is $\Sigma = \{\{P\}, \{\}\}$. We have

$$\begin{aligned} & \lceil P \rceil \Rightarrow (\lceil P \rceil^\frown \lceil P \rceil) \text{ is valid} \\ \text{iff} \quad & \lceil P \rceil \wedge \neg(\lceil P \rceil^\frown \lceil P \rceil) \text{ is not satisfiable} \\ \text{iff} \quad & \mathcal{L}_1(\lceil P \rceil) \cap \mathcal{L}_1(\neg(\lceil P \rceil^\frown \lceil P \rceil)) = \{\} \\ \text{iff} \quad & \{\{P\}^i \mid i \geqslant 1\} \cap \Sigma^* \setminus \{\{P\}^i \mid i \geqslant 2\} = \{\} \end{aligned}$$

which is false as the intersection contains the word $\{P\}$. Thus, the formula $\lceil P \rceil \Rightarrow (\lceil P \rceil^\frown \lceil P \rceil)$ is not valid for discrete time. □

Using this technique we can decide that the formula *Req* from the case study is satisfiable. It is, however, more interesting that the *phase automaton* from [RRH93] can be represented in *RDC* as well. This phase automaton is a formula of DC representing an implementation of the requirements. It is proved in [RRH93] that the phase automaton implies the requirements. Since this implication can be represented in *RDC*, the above algorithm can carry out this proof for a discrete time domain.

5.4. Decidability for Continuous Time

Consider the formula $\lceil P \rceil \Rightarrow (\lceil P \rceil^\frown \lceil P \rceil)$, which is valid for continuous time; but not for discrete time.

Reconsidering the question of its validity for discrete time we have that

$$\lceil P \rceil \Rightarrow (\lceil P \rceil^\frown \lceil P \rceil) \text{ is valid} \quad \text{iff} \quad \mathcal{L}_1(\lceil P \rceil) \subseteq \mathcal{L}_1(\lceil P \rceil^\frown \lceil P \rceil)$$

But because $\{P\} \in \mathcal{L}_1(\lceil P \rceil)$ and $\{P\} \notin \mathcal{L}_1(\lceil P \rceil^\frown \lceil P \rceil) = \{\{P\}^i \mid i \geqslant 2\}$ we have that the inclusion property is not satisfied.

The problem is that a letter, say $\{P\}$, cannot be interpreted as lasting one time unit under a continuous time domain.

However, with a *closure property* it is possible to re-use ideas from the discrete time construction to achieve a decidability result for continuous time.

A language L over alphabet Σ is called *contraction closed* if

$$vaaw \in L \text{ implies } vaw \in L$$

for any $v, w \in \Sigma^*, a \in \Sigma$.

The language $\mathcal{L}_1(\lceil P \rceil^\frown \lceil P \rceil) = \{\{P\}^i | i \geqslant 2\}$ is not contraction closed since $\{P\}\{P\}$ belongs to the language and $\{P\}$ does not belong to the language.

Let $\downarrow L$ denote the contraction closure of L, i.e. the smallest contraction closed set containing L. By a simple construction on finite automata one can establish the following:

Lemma 5.3. If L is regular then so is $\downarrow L$, and $\downarrow L$ can furthermore be constructed from L.

We now construct a regular language $\mathcal{L}_2(\phi)$ from a formula $\phi \in RDC$ in a way similar to the way for discrete time:

$$\begin{aligned}
\mathcal{L}_2(\lceil S \rceil) &= (DNF(S))^+ \\
\mathcal{L}_2(\phi \vee \psi) &= \mathcal{L}_2(\phi) \cup \mathcal{L}_2(\psi) \\
\mathcal{L}_2(\neg\phi) &= \Sigma^* \setminus \mathcal{L}_2(\phi) \\
\mathcal{L}_2(\phi^\frown\psi) &= \downarrow(\mathcal{L}_2(\phi)\mathcal{L}(\psi))
\end{aligned}$$

By an induction proof on the structure of ϕ one can establish:

Lemma 5.4. A formula $\phi \in RDC$ is satisfiable for continuous time iff the language $\mathcal{L}_2(\phi)$ is not empty.

Since $\mathcal{L}_2(\phi)$ is a regular language we have

Theorem 5.2. The satisfiability question in continuous time of DC formulas in *RDC* is decidable.

5.5. Complexity

The efficiency of the above decision procedure depends on constructions on finite automata, and this efficiency is very poor since each negation occurring in the formula may supply an exponent in the complexity. An exponent occurs when a non-deterministic automaton is transformed into a deterministic one when constructing a finite automaton for the complement of a regular language. The authors of [SkS94b] have found that the decision problem is non-elementary. So the worst case is very poor, indeed.

In [SkS94b] the decision procedure is implemented and used to prove the correctness of Fisher's mutual exclusion protocol. The results were not too bad. It took, for example, approximately twelve minutes to verify a formula consisting of 3775 characters on a DECStation 5000-240 with 128 MB of memory.

Furthermore, the situation seems (always) to be that the level of nesting of negations (which cause the poor complexity) is very low for formulas occurring in case studies. The reason for this is that formulas with alternating nesting of negation, e.g. through implication in $(((\phi_1 \Rightarrow \phi_2) \Rightarrow \phi_3) \Rightarrow \phi_4)$ or in $(\neg(\neg((\neg\phi_1)^\frown\phi_2)^\frown\phi_3)^\frown\phi_4)$, are very difficult to comprehend and therefore do not occur in specification examples.

The proof assistant for DC described in [SkS93] also supports the use of this decision procedure.

6. Undecidability

All the disappointing news comes in this section: even for very restricted subsets of DC formulas, it is undecidable whether a formula in the subsets is satisfiable.

The general technique used to show these results is to reduce the halting problem of two-counter machines to the satisfiability of formulas belonging to the subset under consideration. The main results come from [ZHS93]. We give undecidability results for the following three subsets:

6.1. The Subsets $RDC_1(k)$, RDC_2, and RDC_3

The subset $RDC_1(k)$, where $k \in \mathbb{R}$ is a fixed constant, is the subset of propositional DC generated as follows:

1. the formula $\ell = k$ belongs to $RDC_1(k)$,
2. if S is a state expression, then $\lceil S \rceil$ belongs to $RDC_1(k)$, and
3. if ϕ and ψ belong to $RDC_1(k)$, then so do $\neg\phi$, $\phi \vee \psi$, and $\phi^\frown\psi$.

When k is a natural number we have previously seen that it is decidable for discrete time whether a formula of $RDC_1(k)$ is satisfiable.

Since $\ell = 0$ can be expressed by $\neg\lceil 1 \rceil$ we have that $RDC(0)$ is expressible in RDC, and thus the satisfiability question for $RDC(0)$ is decidable for continuous time. If $k < 0$ then $\ell = k$ is false which is expressible in RDC as well. So $k > 0$ is assumed in the following undecidability proof for $RDC_1(k)$.

The subset RDC_2 is the subset of propositional DC generated as follows:

1. if S_1 and S_2 are state expressions, then $\int S_1 = \int S_2$ belongs to RDC_2 and
2. if ϕ and ψ belong to RDC_2, then so do $\neg\phi$, $\phi \vee \psi$, and $\psi^\frown\psi$.

The subset RDC_3 is the subset of DC generated as follows:

1. if S is a state expressions, then $\lceil S \rceil$ belongs to RDC_3,
2. if x is a global variable, then $\ell = x$ belongs to RDC_3, and
3. if ϕ and ψ belong to RDC_3, then so do $\neg\phi$, $\phi \vee \psi$, $\phi^\frown\psi$, and $(\exists x)\phi$, where x is any global variable.

Remark: For any of the different subsets, we shall use standard abbreviations for $\wedge$ and $\Rightarrow$ from propositional logic, and for $\Box$ and $\Diamond$ from IL. $\square$

Counter Machines: The main technique used to obtain these undecidability results is to reduce the halting problem of counter machines to satisfiability of formulas belonging to the subset under consideration. In this section we give a brief and rather informal introduction to counter machines. For a more careful treatment we refer e.g. to [Min67] or [HoU79].

A *two-counter machine* has a current *label* q and two *counters* c_1 and c_2 which can hold arbitrary non-negative integers. A *program* for such a machine is a finite set of labeled *instructions* m_i, i.e. the set of labels is finite.

The only instructions are "increase c_1 by one" (c_1^+) and "test c_1 and decrease by one" (c_1^-), and similarly for c_2. For example, $q_i : c_1^+ \to q_j$ is the instruction at label q_i; it increases c_1 by one and proceeds with the instruction labeled q_j. The instruction $q_i : c_1^- \to q_j, q_k$ tests whether the value of c_1 is zero; if so, the machine proceeds with the instruction labeled q_j. Otherwise, the machine decreases c_1 by one and proceeds with the instruction labeled q_k.

A *configuration* s of a counter machine is a triple $s = (q, n_1, n_2)$ of the current label q and the values $n_1, n_2 \in \mathbb{N}$ of the two counters c_1 and c_2. The configuration (q, n_1, n_2) is *final* if there is no instruction labeled q in the program.

A *computation step* $s \Longrightarrow s'$ transforms a non-final configuration s into a configuration s' as follows (and similarly for c_2):

Instruction	s	$\Longrightarrow s'$
$q : c_1^+ \to q_j$	(q, n_1, n_2)	$\Longrightarrow (q_j, n_1 + 1, n_2)$
$q : c_1^- \to q_j, q_k$	$(q, 0, n_2)$	$\Longrightarrow (q_j, 0, n_2)$
$q : c_1^- \to q_j, q_k$	$(q, n_1 + 1, n_2)$	$\Longrightarrow (q_k, n_1, n_2)$

A *computation* is a (finite or infinite) sequence $\sigma = s_0, s_1, s_2, \ldots$ of configurations, where $s_n \Longrightarrow s_{n+1}$.

We call s_0 the *initial configuration.* In the applications below we will have $s_0 = (q_0, 0, 0)$, where q_0 is a designated *initial label.* We shall make use of the fact that the halting problem for two-counter machines with initial counter values $n_1 = n_2 = 0$ is undecidable [Bir76, p. 78]. This result also holds even if we assume that the programs contain precisely one final label q_{fin}, i.e. q_{fin} is the only label so that no instruction has this as its label. In the following we consider two-counter machines M with the initial configuration $(q_0, 0, 0)$ where

1. $q_0, \cdots, q_m, q_{fin}$ are the labels of M, where q_0 is the initial label and q_{fin} is the final label.
2. c_1 and c_2 are the two counters.
3. $m_1, \cdots, m_l$ are the instructions of M.

6.2. Undecidability of $RDC_1(k)$

We reduce the halting problem for M to satisfiability of a formula in $RDC_1(k)$ (for $k > 0$). The encoding of M uses the following state variables: one state variable Q_i for each label q_i, two state variables C_1 and C_2 to represent the counter values, and two auxiliary state variables B and L, used as delimiters.

Let $\mathcal{Q} = \{Q_0, \ldots, Q_m, Q_{fin}\}$ in the following. The main idea is that a machine configuration (q, n_1, n_2) is encoded on an interval of length $4k$ as follows:

$$|\underbrace{\mathcal{Q}}_{k}|\underbrace{Val_1}_{k}|\underbrace{L}_{k}|\underbrace{Val_2}_{k}| \quad \text{where } Val_j \text{ represents the value of counter } c_j$$

This will be done so that the n-th configuration appears in the interval $[4nk, 4(n+1)k], n \geqslant 0$.

The idea behind the representation of the counter values is the following: Let the value of counter c_i be $n_i \geqslant 0$. Then the interval describing Val_i is required to have the following form:

$$|B|C_i|B|\ldots|B|C_i|B|$$

with n_i occurrences of C_i. Since this interval is required to have the length k, and since there is no bound to the counter value, this idea is based on the denseness of the time domain. This representation was inspired by [ACD90].

The task is to formalize these ideas as a formula in $RDC_1(k)$. In particular, we must construct a formula representing the initial configuration and a formula expressing how the $n+1$'st configuration relates to the n'th configuration. To do so, the following abbreviations of formulas in $RDC_1(k)$ will be useful:

$$
\begin{aligned}
\lceil\rceil &\;\widehat{=}\; \neg\lceil 1\rceil \\
\text{true} &\;\widehat{=}\; \lceil\rceil \vee \lceil 1\rceil \\
\ell < k &\;\widehat{=}\; \neg((\ell = k)^\frown\text{true}) \\
\ell = 2k &\;\widehat{=}\; (\ell = k)^\frown(\ell = k) \\
\ell < 2k &\;\widehat{=}\; \neg((\ell = 2k)^\frown\text{true}) \\
2k \leqslant \ell &\;\widehat{=}\; (\ell = 2k)^\frown\text{true} \\
\ell < 3k &\;\widehat{=}\; \neg((\ell = 2k)^\frown(\ell = k)^\frown\text{true}) \\
\ell = 4k &\;\widehat{=}\; (\ell = 2k)^\frown(\ell = 2k) \\
\ell = 5k &\;\widehat{=}\; (\ell = 4k)^\frown(\ell = k) \\
\lceil S\rceil^k &\;\widehat{=}\; \lceil S\rceil \wedge (\ell = k) \\
\phi \rightsquigarrow \psi &\;\widehat{=}\; \neg(\phi^\frown\neg(\lceil\rceil \vee \psi))
\end{aligned}
$$

The formula $\lceil S\rceil^k$ reads: "S is one for a duration of k", and $\phi \rightsquigarrow \psi$ reads: "if the interval starts with ϕ, it must end immediately with $\lceil\rceil$ or with ψ".

The initial configuration is $(q_0, 0, 0)$, which is represented by the formula:

$$Init \;\widehat{=}\; \lceil Q_0\rceil^k{}^\frown\lceil B\rceil^k{}^\frown\lceil L\rceil^k{}^\frown\lceil B\rceil^k{}^\frown\text{true}$$

State variables must be mutually exclusive:

$$Mutex \;\widehat{=}\; \bigwedge_{P_1 \neq P_2} \Box\neg\lceil P_1 \wedge P_2\rceil \quad \text{where } P_1, P_2 \text{ range over } \mathcal{Q} \cup \{C_1, C_2, B, L\}$$

Certain state expressions will have a periodic appearance since configurations are represented on intervals of length $4k$. Let

$$Per(\phi) \;\widehat{=}\; \Box((\phi^\frown(\ell = 4k)) \;\Rightarrow\; ((\ell = 4k)^\frown\phi))$$

Machine labels, counter values, and the separator L have periodic behaviour:
Periodic $\widehat{=}$

$$Per(\bigvee_{Q_i \in \mathcal{Q}} \lceil Q_i\rceil^k) \wedge Per(\lceil C_1 \vee B\rceil^k) \wedge Per(\lceil L\rceil^k) \wedge Per(\lceil C_2 \vee B\rceil^k)$$

For each instruction m_i of M we give a formula $F(m_i)$, encoding the computation step performed by the instruction.

Consider the machine instruction $m = q_i : c_1^+ \to q_j$. The possible computation steps allowed by m are described by a formula $F(m) \;\widehat{=}\; F_1 \wedge F_2 \wedge F_3 \wedge F_4 \wedge F_5 \wedge F_6$, where each F_i is defined below.

The formula F_1 expresses that q_j is the label of the next configuration:

$$F_1 \;\widehat{=}\; (\lceil Q_i\rceil^k{}^\frown(\ell = 4k)) \;\Rightarrow ((\ell = 4k)\;{}^\frown\lceil Q_j\rceil^k)$$

The formula F_2 copies the C_1's to the same place in the next configuration. To encode it, we use formulas of the form: $\phi \rightsquigarrow \psi$. Here ϕ characterizes certain configurations whose label is Q_i, and ψ fixes part of the next configuration.

$$F_2 \;\widehat{=}\; (\lceil Q_i\rceil^k{}^\frown(\ell < k)^\frown\lceil C_1\rceil^\frown\begin{pmatrix}\lceil C_1\rceil^\frown\text{true} \\ \wedge \\ \ell = 4k\end{pmatrix}) \rightsquigarrow (\lceil C_1\rceil^\frown\text{true})$$

One can copy the B's occurring before a C_1 to the same place in the next configuration using the same technique:

$$F_3 \mathrel{\widehat{=}} (\lceil Q_i \rceil^k \frown (\ell < k) \frown \lceil B \rceil \frown \begin{pmatrix} \lceil B \rceil \frown \lceil C_1 \rceil \frown \text{true} \\ \wedge \\ \ell = 4k \end{pmatrix}) \rightsquigarrow (\lceil B \rceil \frown \text{true})$$

The formulas F_4 and F_5 increase the value of c_1 by replacing the last $\lceil B \rceil$ section of c_1's value with $\lceil B \rceil \frown \lceil C_1 \rceil \frown \lceil B \rceil$ in the next configuration. (F_4 handles the case $n_1 = 0$, and F_5 handles the case $n_1 > 0$.)

$$F_4 \mathrel{\widehat{=}} (\lceil Q_i \rceil^k \frown \lceil B \rceil^k \frown (\ell = 4k)) \Rightarrow (\text{true} \frown (\ell = k \wedge (\lceil B \rceil \frown \lceil C_1 \rceil \frown \lceil B \rceil)))$$

$$F_5 \mathrel{\widehat{=}} \begin{pmatrix} (\lceil Q_i \rceil^k \frown (\ell < k) \frown \lceil C_1 \rceil \frown (\begin{matrix} \lceil B \rceil \frown \lceil L \rceil \frown \text{true} \\ \wedge (\ell = 5k) \end{matrix})) \\ \Rightarrow (\text{true} \frown ((\ell = k) \wedge (\lceil B \rceil \frown \lceil C_1 \rceil \frown \lceil B \rceil \frown \lceil L \rceil))) \end{pmatrix}$$

Note that the beginnings of succeeding L sections are exactly $4k$ apart, and therefore in F_5 the length of the $\lceil B \rceil \frown \lceil C_1 \rceil \frown \lceil B \rceil$ section in the consequent is precisely as long as the last $\lceil B \rceil$ section in the antecedent. Thus, the "effect" of $F_4 \wedge F_5$ is to increase the number of $\lceil C_1 \rceil$ sections by one, as desired.

The formula *Periodic* takes care of copying the L section to the next configuration, and the formula F_6 copies the value of c_2 to the next configuration using the same technique as used above:

$$F_6 \mathrel{\widehat{=}} \begin{matrix} (\lceil Q_i \rceil^k \frown (\ell < 3k) \frown \lceil C_2 \rceil \frown \begin{pmatrix} \lceil C_2 \rceil \frown \text{true} \\ \wedge \\ \ell = 4k \end{pmatrix}) \rightsquigarrow (\lceil C_2 \rceil \frown \text{true}) \\ \wedge \\ (\lceil Q_i \rceil^k \frown (2k \leqslant \ell < 3k) \frown \lceil B \rceil \frown \begin{pmatrix} \lceil B \rceil \frown \text{true} \\ \wedge \\ \ell = 4k \end{pmatrix}) \rightsquigarrow (\lceil B \rceil \frown \text{true}) \end{matrix}$$

The remaining instructions m_i can be encoded as formulas $F(m_i)$ by techniques similar to those already used. Then the entire machine is encoded by:

$$\mathit{Machine} \mathrel{\widehat{=}} \mathit{Mutex} \wedge \mathit{Init} \wedge \mathit{Periodic} \wedge \bigwedge_{m_i} \Box F(m_i)$$

By the construction of formula *Machine* we know that execution of the machine terminates if and only if the ($\mathit{Machine} \wedge \Diamond \lceil Q_{fin} \rceil$) is satisfiable.

Theorem 6.1. The satisfiability in continuous time of DC formulas in $RDC_1(k)$ is undecidable.

Remark: This result depends on the ability to express precisely the length $\ell = k$ of intervals. One would, however, not get a decidable subset if the formula $\ell < k$ is used instead, since $\ell = k$ can be derived from $\ell < k$ as follows[2]:

$$\begin{aligned} \Diamond_{pr} \phi &\mathrel{\widehat{=}} (\lceil 1 \rceil \frown \phi \frown \text{true}) \vee (\text{true} \frown \phi \frown \lceil 1 \rceil) \\ \Box_{pr} \phi &\mathrel{\widehat{=}} \neg \Diamond_{pr} (\neg \phi) \\ \ell = k &\mathrel{\widehat{=}} \neg (\ell < k) \wedge \Box_{pr} (\ell < k) \end{aligned}$$

[2] This was pointed out by Peter Sestoft.

The formula $\Diamond_{pr}\phi$ reads: "for some proper sub-interval: ϕ".

Thus, we cannot achieve a decidable subset by "relaxing the punctuality" from $\ell = k$ to $\ell < k$, analogous to the result discussed in [AFH91]. We do not know whether this is possible when $\ell > k$ is considered instead of $\ell = k$. □

6.3. Undecidability of RDC_2

We reduce the halting problem for two-counter machine M to the satisfiability of a formula in RDC_2. We give a reduction which works both for a discrete and for a continuous time domain. This reduction is simpler than the first one given in [ZHS93]. The following state variables are used in this reduction:

1. two state variables C_i^+ and C_i^- for each counter $c_i, i = 1, 2$.
2. $m+2$ state variables $\mathcal{Q} = \{Q_0, \ldots, Q_m, Q_{fin}\}$ corresponding to the labels of M.

The following abbreviations will be used for state expressions:

$$C^\vee \;\widehat{=}\; C_1^+ \vee C_1^- \vee C_2^+ \vee C_2^-$$
$$C^\wedge \;\widehat{=}\; C_1^+ \wedge C_1^- \wedge C_2^+ \wedge C_2^-$$
$$Q \;\widehat{=}\; Q_0 \vee \cdots \vee Q_m \vee Q_{fin}$$

where $C^\vee$ expresses that at least one of the four counter state variables is one, $C^\wedge$ expresses that all four counter state variables are one, and Q expresses that at least one of the state variables corresponding to the labels is one.

The main idea of the encoding of M is that a *period* is divided into two *sections* $|Q|C|$, where Q is one of the state variables for labels, and C is either one of C_1^+, C_1^-, C_2^+, or C_2^- or their conjunction $C^\wedge$.

A configuration (and a computation) is represented by a sequence of periods:

$$|Q_0|C^\wedge|Q_1'|C|Q_2'|C|\ldots|Q_k'|C|$$

so that $\int C_i^+ = \int C_i^-$ holds for this sequence iff the counter c_i has the value 0 after k computation steps.

In fact, the idea of the encoding is that the value of the counter c_i in the k'th configuration is $\int C_i^+ - \int C_i^-$ where the integration is over the "whole sequence". All sections must have the same size for this idea to work.

To formalize these ideas in RDC_2 we will use the following abbreviations of formulas concerning counter values:

$$\lceil S \rceil \;\widehat{=}\; (\int S = \int 1) \wedge \neg(\int 0 = \int 1)$$
$$Incr_1 \;\widehat{=}\; \lceil C_1^+ \wedge \neg(C_1^- \vee C_2^+ \vee C_2^-) \rceil$$
$$Decr_1 \;\widehat{=}\; \lceil C_1^- \wedge \neg(C_1^+ \vee C_2^+ \vee C_2^-) \rceil$$
$$Incr_2 \;\widehat{=}\; \lceil C_2^+ \wedge \neg(C_2^- \vee C_1^+ \vee C_1^-) \rceil$$
$$Decr_2 \;\widehat{=}\; \lceil C_2^- \wedge \neg(C_2^+ \vee C_1^+ \vee C_1^-) \rceil$$
$$Const \;\widehat{=}\; \lceil C^\wedge \rceil$$

The formula $Incr_1$ expresses that the value of counter c_1 is increased by one by letting C_1^+ be one throughout one section, while the other counter state variables are zero. The formulas $Decr_1, Incr_2, Decr_2$ have similar explanations.

The conjunction $C^{\wedge}$ is used in *Const* to keep the counter values constant from one period to the next (by increasing $\int C_i^+$ as much as $\int C_i^-$).

Furthermore, the following abbreviations of formulas in RDC_2 will be used:

$$\begin{aligned}
&\lceil\rceil && \widehat{=} && \neg\lceil 1\rceil && \\
&\text{true} && \widehat{=} && \lceil\rceil \vee \lceil 1\rceil && \\
&\phi \rightsquigarrow \psi && \widehat{=} && \neg(\phi^\frown\neg(\lceil\rceil \vee \psi)) && \\
&\Diamond_p\phi && \widehat{=} && \phi^\frown\text{true} && \text{“for some prefix interval: } \phi\text{”} \\
&\Box_p\phi && \widehat{=} && \neg(\Diamond_p(\neg\phi)) && \text{“for all prefix intervals: } \phi\text{”}
\end{aligned}$$

Let R and S be two state expressions. By a formula in RDC_2 we describe a sequence of equal size sections of the form:

R	S	R	S	R	S	...	R	S	
1	2	3	4	5	6	...	n	$n+1$	

Let *EqualSizeCover*(R,S) be defined by the formula:

$$\neg\Diamond\lceil R \wedge S\rceil \qquad (a)$$

$$\wedge \begin{pmatrix} \lceil R\rceil^\frown\lceil S\rceil \\ \wedge \\ \int R = \int S \end{pmatrix}^\frown \text{true} \qquad (b)$$

$$\wedge \Box_p((\lceil R\rceil^\frown\lceil S\rceil) \Rightarrow ((\lceil\rceil \vee \lceil R\rceil)^\frown(\int R = \int S))) \qquad (c)$$

$$\wedge \Box((\lceil S\rceil^\frown\lceil\neg S\rceil) \Rightarrow (\lceil S\rceil^\frown\lceil R\rceil)) \qquad (d)$$

$$\wedge \Box((\lceil R\rceil^\frown\lceil S\rceil^\frown\lceil R\rceil) \Rightarrow (\lceil R\rceil^\frown(\lceil\rceil \vee \lceil S\rceil)^\frown \begin{pmatrix} \lceil S\rceil^\frown\lceil R\rceil \\ \wedge \\ \int S = \int R \end{pmatrix})) \qquad (e)$$

$$\wedge \Box((\lceil R\rceil^\frown\lceil\neg R\rceil) \Rightarrow (\lceil R\rceil^\frown\lceil S\rceil)) \qquad (f)$$

$$\wedge \Box((\lceil R\rceil^\frown\lceil S\rceil^\frown\lceil R\rceil^\frown\lceil S\rceil) \Rightarrow (\lceil R\rceil^\frown \begin{pmatrix} \lceil S\rceil^\frown\lceil R\rceil \\ \wedge \\ \int S = \int R \end{pmatrix}^\frown \lceil S\rceil)) \qquad (g)$$

$$\wedge \Box((\lceil S\rceil^\frown\lceil R\rceil^\frown\lceil S\rceil) \Rightarrow (\lceil S\rceil^\frown(\lceil\rceil \vee \lceil R\rceil)^\frown \begin{pmatrix} \lceil R\rceil^\frown\lceil S\rceil \\ \wedge \\ \int R = \int S \end{pmatrix})) \qquad (h)$$

$$\wedge \Box((\lceil S\rceil^\frown\lceil R\rceil^\frown\lceil S\rceil^\frown\lceil R\rceil) \Rightarrow (\lceil S\rceil^\frown \begin{pmatrix} \lceil R\rceil^\frown\lceil S\rceil \\ \wedge \\ \int R = \int S \end{pmatrix}^\frown \lceil R\rceil)) \qquad (i)$$

This formula describes a sequence of equal size sections because: (a) requires that the state expressions R and S are mutually exclusive and (b) requires that there is an initial period $\lceil R\rceil^\frown\lceil S\rceil$ whose $\lceil R\rceil$ and $\lceil S\rceil$ sections are of equal size. Formula (c) ensures that the first $\lceil S\rceil$ is not too long. Thus, the first three formulas characterize the first two sections.

The remaining formulas, i.e. (d) to (i), guarantee that the next two sections have the right size provided that the previous one has: (d) says that an $\lceil R\rceil$

section must follow each $\lceil S \rceil$ section, and (e) ensures that this $\lceil R \rceil$ section is not too long. (f) says that an $\lceil S \rceil$ section will follow each $\lceil R \rceil$ section, and (g) guarantees that each full $\lceil R \rceil$ section is precisely as long as the previous full $\lceil S \rceil$ section. (h) and (i) are similar to (e) and (g).

The following mutual exclusion property is needed:

$$Mutex \mathrel{\widehat{=}} \bigwedge_{P_1 \neq P_2} \neg\Diamond\lceil P_1 \wedge P_2 \rceil \quad \text{where } P_1 \text{ and } P_2 \text{ range over } Q \cup \{C^\vee\}$$

The two-counter machine M is encoded by the formula: $Machine \mathrel{\widehat{=}}$

$$Mutex \wedge EqualSizeCover(Q, C^\vee) \wedge Init \wedge Start(m_0) \wedge \bigwedge_{m_i} \Box_p G(m_i)$$

where $Init$, $Start(m_0)$, and $G(m_i)$ are defined below.

The formula $Init$ expresses that the initial configuration is $(q_0, 0, 0)$:

$$Init \mathrel{\widehat{=}} \begin{pmatrix} \lceil Q_0 \rceil^\frown Const \\ \wedge \\ \int Q_0 = \int C^\wedge \end{pmatrix}^\frown \text{true}$$

The formulas $Start(m_0)$ and $G(m_i)$ are defined below. Here we assume that m_0 is the instruction which has q_0 as its label. Then the formula $Start(m_0)$ describes the first transition of a computation by M and the formula $G(m)$ describes any later transition caused by the instruction m.

In the definition of $Start(m_0)$ below we use that we know from $Init$ that the initial configuration is $(q_0, 0, 0)$.

If m_0 is $q_0 : c_i^+ \to q_j$, then $Start(q_0 : c_i^+ \to q_j) \mathrel{\widehat{=}}$

$$\begin{pmatrix} \lceil Q_0 \rceil^\frown Const \\ \wedge \\ \int Q_0 = \int C^\wedge \end{pmatrix} \rightsquigarrow \begin{pmatrix} \lceil Q_j \rceil \\ \vee \\ \lceil Q_j \rceil^\frown Incr_i \\ \vee \\ \lceil Q_j \rceil^\frown Incr_i{}^\frown \lceil Q \rceil^\frown \text{true} \end{pmatrix}$$

Notice that the right-hand side of $\rightsquigarrow$ demands that $Incr_i$ can only be followed by a $\lceil Q \rceil$ section. This *excludes* for instance a section of the form: $Incr_i{}^\frown Decr_i$. Furthermore, since $\lceil Q_j \rceil$ can only be followed by a section with $Incr_i$ we have due to $Mutex$ that all other state variables corresponding to labels are zero in this $\lceil Q_j \rceil$ section.

If m_0 is $q_0 : c_i^- \to q_j, q_k$, then $Start(q_0 : c_i^- \to q_j, q_k) \mathrel{\widehat{=}}$

$$\begin{pmatrix} \lceil Q_0 \rceil^\frown Const \\ \wedge \\ \int Q_0 = \int C^\wedge \end{pmatrix} \rightsquigarrow \begin{pmatrix} \lceil Q_j \rceil \\ \vee \\ \lceil Q_j \rceil^\frown Const \\ \vee \\ \lceil Q_j \rceil^\frown Const^\frown \lceil Q \rceil^\frown \text{true} \end{pmatrix}$$

Thus, the formula $Init \wedge Start(m_0)$ characterizes the first two configurations of M's computation. In the definition of $G(m)$, which characterizes later configurations, we use that a previous configuration exists for the "actual" transition.

Define $G(q_j : c_i^+ \to q_k) \mathrel{\widehat{=}}$

$$\left(\text{true}^\frown\lceil C^\vee\rceil^\frown\left(\begin{array}{c} \lceil Q_j\rceil^\frown\lceil C^\vee\rceil \\ \wedge \\ \int Q_j = \int C^\vee \end{array} \right) \right) \rightsquigarrow \left(\begin{array}{l} \lceil Q_k\rceil \\ \vee \\ \lceil Q_k\rceil^\frown Incr_i \\ \vee \\ \lceil Q_k\rceil^\frown Incr_i^\frown\lceil Q\rceil^\frown\text{true} \end{array} \right)$$

Define $G(q_j : c_i^- \to q_k, q_l) \mathrel{\widehat{=}}$

$$\left(\begin{array}{c} \left(\text{true}^\frown\lceil C^\vee\rceil^\frown\left(\begin{array}{c} \lceil Q_j\rceil^\frown\lceil C^\vee\rceil \\ \wedge \\ \int Q_j = \int C^\vee \end{array} \right) \right) \rightsquigarrow \left(\begin{array}{l} \lceil Q_k\rceil \\ \vee \\ \lceil Q_k\rceil^\frown Const \\ \vee \\ \lceil Q_k\rceil^\frown Const^\frown\lceil Q\rceil^\frown\text{true} \end{array} \right) \\ \wedge \\ \left(\text{true}^\frown\lceil C^\vee\rceil^\frown\left(\begin{array}{c} \lceil Q_j\rceil^\frown\lceil C^\vee\rceil \\ \wedge \\ \int Q_j = \int C^\vee \end{array} \right) \right) \rightsquigarrow \left(\begin{array}{l} \lceil Q_l\rceil \\ \vee \\ \lceil Q_l\rceil^\frown Decr_i \\ \vee \\ \lceil Q_l\rceil^\frown Decr_i^\frown\lceil Q\rceil^\frown\text{true} \end{array} \right) \end{array} \right)$$

$$\begin{array}{c} \wedge \int C_i^+ = \int C_i^- \\ \wedge \neg(\int C_i^+ = \int C_i^-) \end{array}$$

The first conjunct describes the case where the value of counter c_i is zero, and the second conjunct describes the case of a positive value of counter c_i.

$G(m_i)$ must hold for all *prefix* intervals in *Machine* because the counter value of a configuration is "calculated" from the whole computation leading to the configuration, e.g. c_i has the value 0 in the k'th configuration of the computation:

$$|Q_0|C^\wedge|Q_1|C|Q_2|C|\ldots|Q_k|C|$$

if $\int C_i^+ = \int C_i^-$ holds "for the whole sequence".

It can be proven that if $\mathcal{I}, [a,b] \models_{dc} Machine \wedge \Diamond\lceil Q_{fin}\rceil$, then a terminating computation of M can be constructed, and vice versa, from a terminating computation of M one can construct $\mathcal{I}$ and $[a,b]$ so that $\mathcal{I}, [a,b] \models_{dc} Machine \wedge \Diamond\lceil Q_{fin}\rceil$. Thus, the halting problem for two-counter machines can be reduced to satisfiability of formulas in RDC_2, and we have the following

Theorem 6.2. The satisfiability of DC formulas in RDC_2 is undecidable for discrete as well as for continuous time.

6.4. Undecidability of RDC_3

The halting problem for a two-counter machine M can be reduced to satisfiability of a formula built from $\lceil P\rceil$ and $\ell = x$ using the connectives $\neg$ and $\vee$, the quantifier $\exists x$, and the modality $^\frown$ [ZHS93]. We do not present this reduction, since it illustrates universal quantification better than it does IL and DC.

Theorem 6.3. The satisfiability (in discrete and continuous time) of duration calculus formulas from RDC_3 is undecidable.

7. Related Work

In this section we give references to related work on interval logics and duration calculus, which we know of at present.

7.1. Work on Interval Logic

The first work we know of on interval logic comes from the area of philosophical logic [Hum79, Röp80, Ben83].

In the area of artificial intelligence, the papers [All83, All84] study a theory of action and time. Thirteen binary relations are introduced for the different relationships between two intervals, e.g. i_1 before i_2 and i_1 overlaps i_2. The relationships can be used in first order sentences where one can quantify over intervals, e.g.

$$\forall i_1, i_2, i_3 (i_1 \text{ before } i_2 \ \wedge \ i_2 \text{ before } i_3 \ \Rightarrow \ i_1 \text{ before } i_3)$$

is an axiom in the system. The modality $\frown$ corresponds to a ternary relation of intervals and is not introduced in this work.

In [SMV83] and [HMM83] interval logics (with modalities) are used to describe protocols and hardware components. The modality used in [SMV83] has the form $[I]\alpha$ which reads: "the next time the interval I can be constructed, the formula α will hold for that interval", where intervals are characterized through events. Furthermore, [SMV83] used the modalities "for any suffix interval" and "for some suffix interval".

In [HMM83], the discrete time interval temporal logic is introduced to specify hardware components and to reason about such specifications. These first works initiated a research interest in both practical and theoretical aspects of interval logic, e.g. [Mos83, Mos85a, Mos86, RoP86, HaS86, Mel87, Hal88, Ven90, Ven91, GBJ91, Mos93, Dut95b, Dut95a].

In [Mos83] some axioms and inference rules occur for the chop modality, e.g.

$$\begin{aligned} ((\phi \frown \psi) \frown \varphi) &\Leftrightarrow (\phi \frown (\psi \frown \varphi)) \\ ((\phi \vee \psi) \frown \varphi) &\Leftrightarrow ((\phi \frown \varphi) \vee (\psi \frown \varphi)) \end{aligned}$$

Furthermore, in [HaS86], the complexity of validity and satisfiability problems are studied for a propositional interval logic with modalities which can express all relationships between intervals discussed in [All84]. E.g. the validity problem is shown to be Π_1^1-complete when natural numbers are chosen as time domain, and it is r.e.-hard and belongs to Π_1^2 when real numbers are chosen as time domain. In [Ven90] there is an axiomatization of the logic studied in [HaS86]. Furthermore, [Dut95b, Dut95a, ZhH96a] contain a completeness result for a language based on [HMM83].

Related languages are the Time Interval Calculus [Bri91] and the Sequential Calculus [vKH95], which are based on Tarski's calculus of relations [Tar41], and the MITL logic [AFH91]. MITL is based on the Metric Temporal Logic [Koy90] in the sense that it allows the *until* modality to be indexed with a time interval.

7.2. Work on Duration Calculus

7.2.1. DC Models

Different models are used by designers of real-time systems at different design stages. In order to accommodate all necessary models, different families of functions are considered as state models for DC formulas. Thus, states can be more general than the Boolean valued functions considered in this paper. But although we recognize that different models are useful in different situations, we decline

to develop a universal, but complicated, duration calculus to specify and reason about all the models. The Boolean state model, presented in this paper, is the basis for DC. All other models, which have been introduced, extend this basic model. These models are briefly mentioned below.

Boolean State Model: The basic calculus of DC [ZHR91] axiomatizes state durations, i.e. integrals of Boolean valued functions, for the Boolean state model under the *finite variability* assumption (also called *non-Zeno phenomenon*) of states, i.e. a state can only change its presence and absence finitely many times in a bounded time period. The interval modality chop ($^\frown$) is used by this calculus, which can express safety properties of real-time systems. Formalizations of other models are *conservative* extensions of this calculus.

Boolean State and Event Model: The Boolean state and event model is studied in [ZhL94, ZhH96b], where an event is a Boolean valued δ-function, i.e. a function with value of 1 at *discrete* points. It means that an event is an instant action, which takes place at a time point, iff the Boolean valued δ-function of the event takes value 1 at the point. By relating events to state transitions, this model can be used to refine from state based requirements via mixed state and event specifications to event based specifications or programs.

However, one cannot capture point-values of functions with integrals since the integral of a function at a point is always zero. In order to describe point properties in DC, [ZhL94] and [ZhH96b] propose slightly different approaches.

Integrals $\int P$ are in [ZhL94] replaced by *mean values* $\overline{P}$, where $\overline{P}([b,e]) = e$ if $b = e$, and $\overline{P}([b,e]) = \int_b^e P(t)dt/(e-b)$ if $b < e$. Thus, one can describe point properties of Boolean valued functions by using their mean values in point intervals, and at the same time, integral of P can be defined as $\int P \;\widehat{=}\; (\overline{P} \cdot \ell)$. Additional axioms and rules for reasoning about point properties are developed in [ZhL94]. However, extra atomic formulas and axioms are in [ZhH96b] added to the basic calculus to express and reason about state transitions and events.

Real State Model: A real state model consists of a set of real-valued functions, which describe behaviour of physical components of a software embedded system. A Boolean state represents a property of the real states of the model. Therefore, specifications and reasonings at the state level may have to employ real analysis. [ZRH93] investigates how DC can be combined with real analysis, so that real state models can be specified in the framework of DC. [ZhH96a] furthers this research by formalizing some part of real analysis using expanding modalities.

Dependability: The dependability of a requirement for an implementation can be quantitatively measured by the satisfaction probability of the requirement for this implementation. In respect to the Boolean state model, [LRS93, LRS94] provide designers with a set of rules to reason about whether a given requirement will hold with a sufficiently high probability, given failure probabilities of the components used, where implementation with imperfect components is taken to be a finite automaton with history-independent transition probabilities in discrete time domain. [DaZ94] generalizes this work for continuous time domain.

Finite Divergence Model: The finite variability of states and events stipulates that state transitions and events can happen only finitely many times within a finite time period, and can e.g. be adopted in connection with software systems, where

time progresses discretely. But this assumption may be violated in a software embedded system, where time is continuous. The opposite notion is called *finite divergence* (or *Zeno phenomenon*). [HPZ95] formalizes the finite divergence model by introducing into DC rules to calculate a state duration in a finite divergence model as a limit of its approximations in finite variability model.

Super Dense Computation: A *super dense computation* is a sequence of timeless operations. It is an abstraction of a real-time computation within a context with a grand time granularity. This abstraction has been adopted by digital control system, where the cycle time of a computer may be nanoseconds, while the sampling period of a controller may take seconds. Computation time can, in this case, be neglected, and operations can be considered as timeless actions. [ZhH96b] adapts the chop modality (called *super dense chop*), so that it can map an operation in a grand time space into a time space with finer time granularity.

Expanding Modalities: Only *safety* properties can be expressed with contracting modalities such as $^\frown$ and $\Diamond$. In order to specify *unbounded liveness* and *fairness* properties in DC, [Ska94a, Pan96, EnR94, ZhH96a] present proposals to introduce expanding modalities. [ZhH96a] proves that the left and right neighborhood modalities, $\Diamond_l$ and $\Diamond_r$, are adequate in the sense that the other contracting and expanding modalities suggested in [All84, HMM83, Ven90] can be derived from them in a first order logic with the interval length ℓ. [ZhH96a] establishes a complete first order calculus for $\Diamond_l$, $\Diamond_r$ and ℓ, and demonstrates how expanding modalities are also convenient for formulating notions of real analysis.

Infinite Intervals: The behaviour of real-time systems is, occasionally, assumed to be *infinite*. DC is, however, based on an interval logic of finite intervals. An infinite behaviour is therefore specified in DC as a set of all finite prefixes of the behaviour. Expanding modalities can be used to specify liveness and fairness properties in terms of its finite prefixes. However, an alternative approach is to introduce infinite intervals. An extension of DC, which allows infinite intervals, is described in [ZDL95]. In this extension, a state duration over an infinite interval is determined by a property, which specifies the limit of the state duration over finite intervals. Since the extension includes both finite and infinite intervals, *terminating* and *infinite* system behaviour can easily be expressed.

7.2.2. DC Applications

Case Studies of Software Embedded Systems: DC has been applied to several case studies, e.g. an auto pilot [RaR91], a railway crossing [SRR92], a water level monitor [EKM93], a gas burner [RRH93], an aircraft traffic controller [Ina94], a production cell [PeR94], a motor-load control system [YWZ94], an inverted pendulum [WCH96], and a hydraulic actuator system [RRH95]. A case study to formalize and synthesize an optimal design of a double-tank control system is conducted in [HeZ95].

Real-time Specification and Verification: DC has been used to define real-time semantics of other languages, e.g. CSP-like languages [ZHR92, HeB92, ScO95, Sch95, ZhH96b]. It is in [ZHR92] emphasized that some processes may be executed on the same processor. In [ZhH96b], it is assumed that assignments and message passing take no time and thus can constitute a super dense computation.

In [HOS93], DC is used to give semantics to two different formalisms to describe reactive systems, where one supports a global view of systems and the other supports a CSP-like view of systems. In [RRH93], a CSP semantic domain is lifted to DC formulas by regarding traces and refusals as functions of time. [ZWR96] gives semantics to a CSP-like language with continuous variables, which can be used to describe software embedded systems.

In [MGH96] and [PRS95], DC is used to define real-time semantics for SDL and Esterel, respectively. A DC semantics is, in [Die96], proposed for a graphical language called Constraint Diagrams. In [HRS96], a formal meaning of Fault Trees is given using DC. In [HZS92], DC is used to specify and reason about real-time properties of circuits. In [Ris92], the correctness of Fischer's mutual exclusion protocol is proven in DC, and DC is in [ZhZ94] used to specify and verify the deadline driven scheduler. Furthermore, several well-known real-time schedulers are specified in [ChD95].

Refinement of DC Specifications: Refinement laws towards DC *implementables*, which are formulas of a restricted form which can express properties such as timed progress and stability, are considered in [MRR93]. A full exposition of these ideas is given in the monograph [Rav95], which e.g. also contains a study of how to ensure that a set of implementables is *feasible*, i.e. that it is consistent and that any finite observation can be extended in time. Techniques are, in [ORS96, ScO95, Sch95], developed to refine a feasible set of DC implementables via a mixed specification and programming language into an executable program. Refinement of DC implementables into automata is considered in [Sch94, DaW94, Kää95a, Kää95b]. An approach to refining DC specifications into programs in the style of the Hoare logic is considered in [XuH95, XuY96]. In [Lak96], it is shown how Implementables [Rav95] can be encoded in a decidable fragment of a logic called *Duration Interval Logic*.

7.2.3. DC Tools

The research of DC tools include developments of *complete* calculi for interval modalities and state durations, and *decision* procedures and *model checking* algorithms for DC subsets. It is proved that the interval logic [ZhH96a], with modalities $\diamond_l$ and $\diamond_r$, and the interval length ℓ, is complete, and the calculi for integral and mean value are *relatively* complete [HaZ92, Li93]. The decidable subsets of DC are discovered by [ZHS93, Han94, Li93]. Decidability and expressiveness results are presented in [Pan95] for extensions of the mean value calculus which includes expanding modalities and quantifications over state variables. Furthermore, in [PaR95] there is study of mean value calculus extended with fixed point operators. Various models are investigated in [Frä96] which restrict the variability of states. Results on decidability and undecidability of formulas are developed wrt. these restricted models.

In order to check whether state transition sequences of a real-time automaton satisfy a linear inequality of state durations, [KPS93, ZZY94, LiH96] develop algorithms which employ techniques from linear and integer programming.

A proof assistant for DC [SkS94a, Ska94b] is developed as an extension of PVS [OSR93]. A decision procedure [ZHS93] is incorporated in this proof assistant. For example the soundness proof for the induction rules for DC [HaZ92] is checked with this proof assistant. Furthermore, several proofs done in case studies are checked using the DC extension of PVS [Ska94b], e.g. [ZHR91] and

[SRR92]. In these applications of the proof assistant errors in the original proofs were spotted. There is also a proof assistant developed in UNU/IIST [MXW96].

7.3. Other Languages with a Duration Notion

In [MaP93], there is a discussion of how the temporal logic of [MaP92] can be extended by a *duration function* $\int p$ (and a special real-time clock variable T). $\int p$ is a measure of the accumulated time in which p has been true in the interval $[0, T]$.

Another idea is presented in [Lam93], where a construct is introduced so that the duration of some state variable p is related to another state variable x and the actual time, which is called *now*, in such a way that the value of $x(now + t)$ for $t \geqslant 0$ equals the duration of p in the interval $[now, now + t]$.

In [LaH94], a duration term $\int^{\tau} p$ is introduced into the metric temporal logic of [Koy92]. The duration term $\int^{\tau} p$ denotes at a given time t the duration of p in the interval $[t, t + \tau]$. This work is supported by a theorem prover [Hoo94].

In [BES93] there is another logic which can express duration constraints.

In automata based formalism, *integrator variable* is used to measure the accumulated time an automaton has spent in certain states, e.g. [KPS93, ACH93, BER94, KHM94, SiM94].

Acknowledgment

The research of DC [ZHR91] was introduced in the ProCoS (ESPRIT BRA 3104 and 7071) project [Bjø89], when the work of the project was to investigate formal techniques for designing safety critical real-time systems. In a project case study of a gas burner system [SRR90], it was realized that state duration had not yet been well studied as an essential measurement of real-time behaviour of computing systems. A research of a logic for state durations was therefore initiated in 1990. The first paper was published in 1991, and dozens of papers have been published since then. We would like to thank all the colleagues in the project for providing a very stimulating environment.

Special thanks go to all the friends with whom we have written joint papers: Chen Zongji, Martin Fränzle, Jens C. Godskesen, He Weidong, Tony Hoare, Dang Van Hung, Burghard von Karger, Li Xiaoshan, Liu Zhiming, Simon Mørk, Marcus Müller-Olm, Ernst-Rüdiger Olderog, Paritosh K. Pandya, Anders P. Ravn, Hans Rischel, Michael Schenke, Peter Sestoft, Robin Sharp, Jens Ulrik Skakkebæk, Jørgen Staunstrup, Erling Vagn Sørensen, Wang Ji, Belawati H. Widjaja, Yang Lu, Yu Xinyao, Yu Huiqun, Zhang Jingzhong, and Zheng Yuhua.

Furthermore, we would like to thank our colleagues: Henrik Reif Andersen, Dines Bjørner, Jonathan Bowen, Steven Brien, Chris George, He Jifeng, Tony Hoare, Hans Henrik Løvengreen, Søren Prehn, Anders P. Ravn, Hans Rischel, Peter Sestoft, Erling Vagn Sørensen, and Xu Qiwen at the Department of Computer Science, DTU, at PRG, Oxford University, and at UNU/IIST for many discussions, suggestions, corrections, and criticism and other things which are helpful in the daily work.

The writing of this work started when the first author was visiting the Semantics Group at Oldenburg University, Germany. The great help, support and encouragement from Ernst-Rüdiger Olderog during this visit are greatly

appreciated. The first handwritten notes were used in a course on "real-time systems" at Oldenburg University, and we would like to thank Regine Bauer, Jürgen Bohn, Cheryl Dietz, Stephan Kleuker, Stephan Rössig, Michael Schenke from the Semantics Group, and the other attendants in this course for very useful feedback. Likewise we thank for feedback given to us in connection with similar courses at the Technical University of Denmark, UNU/IIST, Railways Computer Center (Beijing), Asian Institute of Technology (Bangkok), Tata Research Design & Development Center (Pune), University of Indonesia, University of Macau, University of the Philippines, and the Institute of Cybernetics (Tallinn).

This research has been funded by several sources. We greatly appreciate funding from: (a) the Commission of the European Communities (CEC) under the projects: ESPRIT BRA 3104 and 7071, the KIT 010, and the HCM project ERB4001GT920879, (b) the Danish Natural Science Research Council, (c) the Danish Technical Research Council under the projects RapID and Codesign, and (d) UNU/IIST and the Chinese State Science and Technology Committee under the project DeTfoRS.

References

[All83] Allen, J. F.: Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[All84] Allen, J.F.: Towards a General Theory of Action and Time. *Artificial Intelligence*, 23:123–154, 1984.

[ACD90] Alur, R., Courcoubetis, C. and Dill, D.: Model-Checking for Real-Time Systems. In *Fifth Annual IEEE Symp. on Logic in Computer Science*, IEEE Press, 1990, pages 414–425.

[ACH93] Alur, R., Courcoubetis, C., Henzinger, T. and Ho, P-H.: Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems. In [GNR93], pages 209–229.

[AlD92] Alur, R. and Dill, D.: The Theory of Timed Automata. In *Real-Time: Theory in Practice*, LNCS 600, Springer-Verlag, 1992, pages 45–73.

[AFH91] Alur, R., Feder, T. and Henzinger, T.: The Benefits of Relaxing Punctuality. In *Tenth Annual ACM Symposium on Principles of Distributed Computing*, ACM Press, 1991, pages 139–152.

[Ben83] Benthem, J. F. A. K. van: *The Logic of Time*. D. Reidel publishing Company, 1983.

[Bir76] Bird, R.: *Programs and Machines*. John Wiley & Sons, 1976.

[Bjø89] Bjørner, D. et al.: A ProCoS Project Description: ESPRIT BRA 3104. *Bulletin of the EATCS*, (39):60–73, 1989.

[BER94] Bouajjani, A., Echahed, R. and Robbana, R.: Verifying Invariance Properties of Timed Systems with Duration Variables. In [LRV94], pages 193–210.

[BES93] Bouajjani, A., Echahed, R. and Sifakis, J.: On Model Checking for Real-Time Properties with Durations. In *Eights Annual IEEE Symp. on Logic in Computer Science*, IEEE Press, 1993, pages 147–159.

[Bri91] Brien, S.: *A time Interval Calculus*. Master's thesis, Programming Research Group, Computing Laboratory, Oxford University, 1991.

[ChD95] Chan, P. and Dang, Van Hung: Duration Calculus Specification of Scheduling for Tasks with Shared Resources. In *Asian Computing Science Conference 1995*, LNCS 1023, Springer-Verlag, 1995, pages 365–380.

[DaZ94] Dang, Van Hung and Zhou, Chaochen: *Probabilistic Duration Calculus for Continuous Time*. UNU/IIST Report No. 25, UNU/IIST, International Institute for Software Technology, P.O. Box 3058, Macau, 1994.

[DaW94] Dang, Van Hung and Wang, Ji: *On Design of Hybrid Control Systems using I/O Automata Models*. UNU/IIST Report No. 35, UNU/IIST, International Institute for Software Technology, P.O. Box 3058, Macau, 1994.

[Die96] Dietz, C.: *Graphical Formalization of Real-Time Requirements*. Technical report, Fachbereich Informatik, Oldenburg University, 1996.

[Dut95a] Dutertre, B.: Complete Proof Systems for First Order Interval Temporal Logic. In *Tenth Annual IEEE Symp. on Logic in Computer Science*, IEEE Press, 1995, pages 36–43.

[Dut95b] Dutertre, B.: *On First Order Interval Temporal Logic.* Report no. CSD-TR-94-3, Department of Computer Science, Royal Holloway, University of London, Egham, Surrey TW20 0EX, England, 1995.

[EKM93] Engel, M., Kubica, M., Madey, J., Parnas, D. L., Ravn, A. P. and Schouwen, A. J. van: A Formal Approach to Computer Systems Requirements Documentation. In [GNR93], pages 452–474.

[EnR94] Engel, M. and Rischel, H.: *Dagstuhl-Seminar Specification Problem - a Duration Calculus Solution.* Department of Computer Science, Technical University of Denmark – Private Communication, 1993.

[Frä96] Fränzle, M.: *Controller Design from Temporal Logic: Undecidability need not matter,* PhD thesis, Kiel University, submitted December 1996.

[GBJ91] Goswami, A., Bell, Michael and Joseph, Mathai: ISL: An Interval Logic for the Specification of Real-Time Programs. In *Formal Techniques in Real-Time and Fault-Tolerant Systems,* LNCS 571, Springer-Verlag, 1991, pages 1–20.

[GNR93] Grossman, R. L., Nerode, A., Ravn, A. P. and Rischel, H. (eds).: *Hybrid Systems,* LNCS 736, Springer-Verlag, 1993.

[Hal88] Hale, R.: *Programming in Temporal Logic.* PhD thesis, Computing Laboratory, Cambridge University, UK, 1988.

[HMM83] Halpern, J., Moskowski, B. and Manna, Z.: A Hardware Semantics based on Temporal Intervals. In *ICALP'83,* LNCS 154, Springer-Verlag, 1983, pages 278–291.

[HaS86] Halpern, J. Y. and Shoham, Y.: A Propositional Modal Logic of Time Intervals. In *Symposium on Logic in Computer Science,* IEEE Press, 1986, pages 279–292.

[HRS96] K. M. Hansen, Ravn, A. P. and Stavridou, V.: *From Safety Analysis to Formal Specification.* Technical report, Department of Information Technology, Technical University of Denmark, 1996.

[Han94] Hansen, M. R.: Model-Checking Discrete Duration Calculus. *Formal Aspects of Computing,* 6A:826–845, 1994.

[HaZ92] Hansen, M. R. and Zhou, Chaochen: Semantics and Completeness of Duration Calculus. In *Real-Time: Theory in Practice,* LNCS 600, Springer-Verlag, 1992, pages 209–225.

[HZS92] Hansen, M. R., Zhou, Chaochen and Staunstrup, J.: A Real-Time Duration Semantics for Circuits. In *TAU'92: 1992 Workshop on Timing Issues in the Specification and Synthesis of Digital Systems, Princeton Univ., NJ.* ACM/SIGDA, 1992.

[HOS93] Hansen, M. R., Olderog, E.-R., Schenke, M., Fränzle, M., Karger, B. von, Müller-Olm, M. and Rischel, H.: *A Duration Semantics for Real-Time Reactive Systems.* Report no. OLD MRH 1/1, ProCoS II, ESPRIT BRA 7071, Oldenburg University, Germany, 1993.

[HPZ95] Hansen, M. R., Pandya, P. K. and Zhou, Chaochen: Finite Divergence. *Theoretical Computer Science,* 138:113–139, 1995.

[HLP90] Harel, E., Lichtenstein, O. and Pnueli, A.: Explicit Clock Temporal Logic. In *Symp. on Logic in Comp. Sci.,* IEEE press, 1990, pages 402–413.

[He94] He, Jifeng: From CSP to Hybrid Systems. In *A Classical Mind: Essays in Honour of C.A.R. Hoare,* Prentice Hall, 1994, pages 171–190.

[HeB92] He, Jifeng and Bowen, J.: Time Interval Semantics and Implementation of a Real-Time Programming Language. In *1992 Euromicro Workshop on Real-Time Systems.* IEEE Press, 1992.

[HeZ95] He, Weidong and Zhou, Chaochen: A Case Study of Optimization. *The Computer Journal,* 38(9):734–746, 1995.

[Hoo94] Hooman, J.: Correctness of Real Time Systems by Construction. In [LRV94], pages 19–40.

[HoU79] Hopcroft, J. E. and Ullman, J. D.: *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[HuC68] Hughes, G. E. and Crestwell, M. J.: *An Introduction to Modal Logic.* Routledge, 1968.

[Hum79] Humberstone, I. L.: Interval Semantics for Tense Logics: Some Remarks. *Journal of Philosophical Logic,* 8:171–196, 1979.

[Ina94] Inal, R.: Modular Specification of Real-Time Systems. In *1994 Euromicro Workshop on Real-Time Systems,* IEEE Press, 1994.

[JaM86] Jahanian, F. and Mok, A. K-L.: Safety Analysis of Timing Properties in Real-Time Systems. *IEEE Trans. SE.,* 12(9), 1986, pages 890–904.

[KHM94] Kapur, A., Henzinger, T., Manna, Z. and Pnueli, A.: Proving Safety Properties of Hybrid Systems. In [LRV94], pages 431–454.

[KPS93] Kesten, Y., Pnueli, A., Sifakis, J. and Yovine, S.: Integration Graphs: A Class of Decidable Hybrid Systems. In [GNR93], pages 179–208.

[Koy90] Koymans, R.: Specifying Real-Time Properties with Metric Temporal Logic. *Real-Time Systems, Kluwer Academic Publishers*, 2(4):255–299, 1990.

[Koy92] Koymans, R.: *Specifying Message Passing and Time-Critical Systems with Temporal Logic.* LNCS 651, Springer-Verlag, 1992.

[Kää95a] Kääramees, M.: Transforming Designs Towards Implementations. In *1995 Euromicro Workshop on Real-Time Systems*, IEEE Press, 1995, pages 197–204.

[Kää95b] Kääramees, M.: Transformation of Duration Calculus Specifications to DisCo Language. Master's thesis, Automation and Systems Engineering, Tallinn Technical University, Estonia, 1995.

[Lak96] Lakhnech, Y.: *Specification and Verification of Hybrid and Real-Time Systems.* PhD thesis, Kiel University, 1996.

[LaH94] Lakhnech, Y. and Hooman, J.: Reasoning about Durations in Metric Temporal Logic. In [LRV94], pages 488–510.

[Lam93] Lamport, L.: Hybrid Systems in TLA^+. In [GNR93], pages 77–102.

[LRV94] Langmack, H., Roever, W.-P. de and Vytopil, J. (eds).: *Formal Techniques in Real-Time and Fault-Tolerant Systems*, LNCS 863, Springer-Verlag, 1994.

[Li93] Li, Xiaoshan: *A Mean Value Calculus.* PhD thesis, Software Institute, Academia Sinica, 1993.

[LiH96] Li, Xuandong and Dang, Van Hung: Checking Linear Duration Invariants by Linear Programming. In *Concurrency and Parallelism, Programming, Network and Security*, LNCS 1179, Springer-Verlag, 1996.

[LRS93] Liu, Zhiming, Ravn, A. P., Sørensen, E. V. and Zhou, Chaochen: A Probabilistic Duration Calculus. In *Dependable Computing and Fault-Tolerant Systems Vol. 7: Responsive Computer Systems*, Springer-Verlag, Wien, New York, 1993, pages 30–52.

[LRS94] Liu, Zhiming, Ravn, A. P., Sørensen, E. V. and Zhou, Chaochen: Towards a Calculus of Systems Dependability. *High Integrity Systems*, 1(1):49–75, 1994.

[MaP92] Manna, Z. and Pnueli, A.: *The Temporal Logic of Reactive and Concurrent Systems.* Springer Verlag, 1992.

[MaP93] Manna, Z. and Pnueli, A.: Verifying Hybrid Systems. In [GNR93], pages 4–35.

[MXW96] Mao, Xiaoguang, Xu, Qiwen and Wang, Ji: *Towards a Proof Assistant for Interval Logics.* UNU/IIST Report No. 77, UNU/IIST, International Institute for Software Technology, P.O. Box 3058, Macau, 1996.

[MRR93] Masiero, P. C., Ravn, A. P. and Rischel, H.: *Refinement of Real-Time Specifications.* Report no. ID/DTH PCM 1/1, ProCoS II, ESPRIT BRA 7071, Department of Computer Science, Technical University of Denmark, 1993.

[Mel87] Melliar-Smith, P. M.: Extending Interval Logic to Real Time Systems. In *Temporal Logic in Specification*, LNCS 398, Springer-Verlag, 1987, pages 224–242.

[Min67] Minsky, M. L.: *Computation: Finite and Infinite Machines.* Prentice Hall, Englewood Cliffs, N.J., 1967.

[Mos83] Moszkowski, B.: *Reasoning about Digital Circuits.* PhD thesis, Stanford University, 1983.

[Mos85a] Moszkowski, B.: A Temporal Logic for Multilevel Reasoning about Hardware. *IEEE Computer*, 18(2):10–19, 1985.

[Mos85b] Moszkowski, B.: Compositional Reasoning about Projected and Infinite Time. In *First IEEE Intl. Conf. on Engineering of Complex Computer Systems*, IEEE press, 1995.

[Mos86] Moszkowski, B.: *Executing Temporal Logic Programs.* Cambridge University Press, Cambridge, UK, 1986.

[Mos93] Moszkowski, B.: Some Very Compositional Temporal Properties. In *Programming Concepts, Methods and Calculi, IFIP Transactions, Vol. A-56*, North-Holland, 1994, pages 307–326.

[MGH96] Mørk, S., Godskesen, J. C., Hansen, M. R. and Sharp, R.: A Timed Semantics for SDL. In *Formal Description Techniques IX: Theory, application and tools*, Chapman & Hall, 1996, pages 295–309.

[NOS93] Nicollin, X., Olivero, A., Sifakis, J. and Yovine, S.: An Approach to the Description and Analysis of Hybrid Systems. In [GNR93], pages 149–178.

[ORS96] Olderog, E-R., Ravn, A. P. and Skakkebæk, J. U.: Refining System Requirements to Program Specifications. In *Formal Methods in Real-Time Systems*, Trends in Software-Engineering, Chapter 5, Wiley, 1996, pages 107–134.

[OSR93] Owre, S., Shankar, N. and Rushby, J.: *Users Guide for the PVS Specification and Verification System, Language, and Proof Checker (beta release)* (three volumes). Technical report, Computer Science Laboratory, SRI International, Menlo Park, CA 94025, USA, 1993.

[Pan95] Pandya, P. K.: *Some Extensions to Propositional Mean Value Calculus: Expressiveness*

and Decidability. Report no. TCS-95/9, Computer Science Group, TIFR, Bombay, 1995. To appear in proc. of CSL'95.

[Pan96] Pandya, P. K.: Weak Chop Inverses and Liveness in Duration Calculus. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*, LNCS 1135, Springer-Verlag, 1996, pages 148–167.

[PaR95] Pandya, P. K. and Ramakrishna, Y.S.: *A Recursive Mean Value Calculus*. Report no. TCS-95/3, Computer Science Group, TIFR, Bombay, 1995.

[PRS95] Pandya, P. K., Ramakrishna, Y. S. and Shyamasundar, R. K.: A Compositional Semantics of Esterel in Duration Calculus. In *Proc. Second AMAST workshop on Real-Time systems: Models and Proofs*, Bordeux, June, 1995.

[PeR94] Petersen, J. L. and Rischel, H.: Formalizing Requirements and Design for a Production Cell System. In *Symposium ADPM '94: Automatisation des Processus Mixtes: Les Systemes Dynamiques Hybrides*, Belgian Institute of Automatic Control, IBRA, 1994, pages 37–46.

[RRH95] Ravn, A. P., Rischel, H., Holdgaard, M., Eriksen, T. J., Conrad, F. and Andersen, T. O.: Hybrid Control of a Robot - a case study. In *Hybrid Systems II*, LNCS 999, Springer-Verlag, 1995, pages 391–404.

[Rav95] Ravn, A. P.: *Design of Embedded Real-Time Computing Systems*. Department of Computer Science, Technical University of Denmark, DK-2800 Lyngby, Denmark, Doctoral Dissertation, ID-TR: 1995-170, 1995.

[RaR91] Ravn, A. P. and Rischel, H.: Requirements Capture for embedded real-time systems. In *Proceedings of IMACS-MCTS'91 Symposium on Modelling and Control of Technological Systems, Villeneuve d'Ascq, France, May 7-10*, volume 2, IMACS, 1991, pages 147–152.

[RRH93] Ravn, A. P., Rischel, H. and Hansen, K. M.: Specifying and Verifying Requirements of Real-Time Systems. *IEEE Trans. SE.*, 19(1):41-55, 1993.

[Ris92] Rischel, H.: *A Duration Calculus Proof of Fischer's Mutual Exclusion Protocol*. Report no. DTH HR 4/1, ProCoS II, ESPRIT BRA 7071, Department of Computer Science, Technical University Of Denmark, 1992.

[RoP86] Rosner, R. and Pnueli, A.: A Choppy Logic. In *First Annual IEEE Symp. on Logic in Computer Science*, IEEE Press, 1986, pages 306–313.

[Röp80] Röper, P.: Intervals and Tenses. *Journal of Philosophical Logic*, 9:451–469, 1980.

[Sch94] Schenke, M.: Specification and Transformation of Reactive Systems with Time Restrictions and Concurrency. In [LRV94], pages 605–620.

[Sch95] Schenke, M.: *Requirements to Programs: A Development Methodology for Real Time Systems, Part 2*. Technical report, Fachbereich Informatik, Universität Oldenburg, 1995. (To appear in Acta Informatica.)

[ScO95] Schenke, M. and Olderog, E.-R.: *Requirements to Programs: A Development Methodology for Real Time Systems, Part 1*. Technical report, Fachbereich Informatik, Universität Oldenburg, 1995.

[SMV83] Schwartz, R. L., Melliar-Smith, P. M. and Vogt, F. H.: An Interval Logic for Higher-Level Temporal Reasoning. In *Second Annual ACM Symposium on Principles of Distributed Computing*, ACM, 1983, pages 173–186.

[SiM94] Sipma, H. B. and Manna, Z.: Specification and Verification of Controlled Systems. In [LRV94], pages 641–659.

[Ska94a] Skakkebæk, J. U.: Liveness and Fairness in Duration Calculus. In *CONCUR'94: Concurrency Theory*, LNCS 836, Springer Verlag, 1994, pages 283–298.

[Ska94b] Skakkebæk, J. U.: *A Verification Assistant for a Real-Time Logic*. PhD thesis, Department of Computer Science, Technical University of Denmark, 1994.

[SRR92] Skakkebæk, J. U., Ravn, A. P., Rischel, H. and Zhou, Chaochen: Specification of Embedded, Real-Time Systems. In *Proceedings of 1992 Euromicro Workshop on Real-Time Systems*, IEEE Press, 1992.

[SkS93] Skakkebæk, J. U. and Shankar, N.: *A Duration Calculus Proof Checker: Using PVS as a Semantic Framework*. Report no. SRI-CSL-93-10, Computer Science Laboratory, SRI International, Menlo Park, CA 94025, USA, 1993.

[SkS94a] Skakkebæk, J. U. and Shankar, N.: Towards a Duration Calculus Proof Assistant in PVS. In [LRV94], pages 660–679.

[SkS94b] Skakkebæk, J. U. and Sestoft, P.: *Checking Validity of Duration Calculus Formulas*. Report no. ID/DTH JUS 3/1, ProCoS II, ESPRIT BRA 7071, Department of Computer Science, Technical University of Denmark, 1994.

[SRR90] Sørensen, E. V., Ravn, A. P. and Rischel, H.: *Control Program for a Gas Burner: Part 1: Informal Requirements, ProCoS Case Study 1*. Report no. ID/DTH EVS2,

ProCoS, ESPRIT BRA 3104, Department of Computer Science, Technical University of Denmark, 1990.

[Tar41] Tarski, A.: On the Calculus of Relations. *Journal of Symbolic Logic*, 6(3):73–88, 1941.

[Ven90] Venema, Y.: Expressiveness and Completeness of an Interval Tense Logic. *Notre Dame Journal of Formal Logic*, 31(4):529–547, 1990.

[Ven91] Venema, Y.: A Modal Logic for Chopping Intervals. *J. Logic Computat.*, 1(4):453–476, 1991.

[vKH95] Karger, B. von and Hoare, C. A. R.: Sequential Calculus. *Information Processing Letters*, 53:123–130, 1995.

[WCH96] Widjaja, Belawati H., Chen, Zongji, He, Weidong and Zhou, Chaochen: A Cooperative Design for Hybrid Control Systems. In *Logic and Software Engineering, International Workshop in Honor of Chih-Sung Tang*, World Scientific, 1996, pages 127-150.

[XuY96] Xu, Qiwen and Yang, Zengyu: Derivation of Control Programs: a Heating System. Presented at the *4th International Conference on Hybrid Systems*, Ithaca, NY, USA, 1996.

[XuH95] Xu, Quiwen and He, Weidong: Hierarchical Design of a Chemical Concentration Control System. In *Hybrid Systems III*, LNCS 1066, Springer-Verlag, 1996, pages 270–281.

[YWZ94] Yu, Xinyao, Wang, Ji, Zhou, Chaochen and Pandya, Paritosh K.: Formal Design of Hybrid Systems. In [LRV94], pages 738–755.

[ZhZ94] Zheng, Yuhua and Zhou, Chaochen: A Formal Proof of the Deadline Driven Scheduler. In [LRV94], pages 756–775.

[ZhH96a] Zhou, Chaochen and Hansen, M. R.: *An Adequate First Order Interval Logic*. UNU/IIST Report No. 91, UNU/IIST, International Institute for Software Technology, P.O. Box 3058, Macau, December 1996.

[ZhH96b] Zhou, Chaochen and Hansen, M. R.: Chopping a Point. In *BCS-FACS 7th refinement workshop*, Electronic Workshops in Computing, Springer-Verlag, 1996.

[ZHR92] Zhou, Chaochen, Hansen, M. R., Ravn, A. P. and Rischel, H.: Duration Specifications for Shared Processors. In *Symposium on Formal Techniques in Real-Time and Fault Tolerant Systems*, LNCS 571, Springer-Verlag, 1991, pages 21–32.

[ZHS93] Zhou, Chaochen, Hansen, M.R. and Sestoft, P.: Decidability and Undecidability Results for Duration Calculus. In *STACS'93*, LNCS 665, Springer-Verlag, 1993, pages 58–68.

[ZHR91] Zhou, Chaochen, Hoare, C. A. R. and Ravn, A. P.: A Calculus of Durations. *Information Processing Letters*, 40(5):269–276, 1991.

[ZDL95] Zhou, Chaochen, Dang, Van Hung and Li, Xiaoshan: A Duration Calculus with Infinite Intervals. In *Fundamentals of Computation Theory*, LNCS 965, Springer-Verlag, 1995, pages 16–41.

[ZWR96] Zhou, Chaochen, Wang, Ji and Ravn, Anders P.: A Formal Description of Hybrid Systems. In *Hybrid Systems III*, LNCS 1066, Springer-Verlag, 1996, pages 511–530.

[ZZY94] Zhou, Chaochen, Zhang, Jingzhong, Yang, Lu and Li, Xiaoshan: Linear Duration Invariants. In [LRV94], pages 86–109.

[ZRH93] Zhou, Chaochen, Ravn, A. P. and Hansen, M. R.: An Extended Duration Calculus for Hybrid Systems. In [GNR93], pages 36–59.

[ZhL94] Zhou, Chaochen and Li, Xiaoshan: A Mean Value Calculus of Durations. In *A Classical Mind: Essays in Honour of C. A. R. Hoare*, Prentice Hall, 1994, pages 431–451.

Received August 1995
Accepted in revised form January 1997 by C. B. Jones