A Theory of Duration Calculus with Application

Michael R. Hansen[1]* and Dang Van Hung[2]

[1] Informatics and Math. Modelling, Technical University of Denmark
Ricard Petersens Plads, DK-2800 Lyngby, Denmark
mrh@imm.dtu.dk
[2] United Nations University, Institute of Software Technology
Casa Silva Mendes, Est. do Engenheiro Trigo No. 4, Macao
dvh@iist.unu.edu

Abstract. In this chapter we will present selected central elements in the theory of Duration Calculus and we will give examples of applications. The chapter will cover syntax, semantics and proof system for the basic logic. Furthermore, results on decidability, undecidability and model-checking will be presented. A few extensions of the basic calculus will be described, in particular, Hybrid Duration Calculus and Duration Calculus with iterations. Furthermore, a case study: the bi-phase mark protocol, is presented. We will not attempt to be exhaustive in our coverage of topics; but we will provide references for further study.

Keywords: Real-time systems, metric-time temporal logic, duration calculus, decidability, model-checking, application

1 Introduction to Duration Calculus

In this chapter we will introduce *Durations Calculus* (abbreviated DC) [72], present central elements of the theory, and show examples of applications. The aim is not to make a comprehensive presentation of the logic; but rather to cover central parts of the logic in a way that readers afterwards can study research papers on the topic. We refer to the monograph [70] for a thorough introduction to DC. The chapter [25] on Duration Calculus in [6] contains an introduction to DC and states major results without proofs, and addresses approaches to modelling real-time systems.

1.1 Background

Duration Calculus is an interval logic which was introduced by Zhou Chaochen, C.A.R. Hoare and A.P. Ravn [72] in 1991, in connection this the ProCoS I project (Provably Correct Systems), ESPRIT BRA 3104, 1989 – 1991, see [5]. In that project, formal techniques for the construction of provably correct systems were studied. Case studies of embedded real-time system, e.g. the Gas Burner

* This work has been partially funded by The Danish Council for Strategic Research under project **MoDES**, the Danish National Advanced Technology Foundation under project **DaNES**, and ARTIST2 (IST-004527).

case study by E.V. Sørensen, H. Rischel and A.P. Ravn, showed that there were a collection of properties which could not be expressed using the specification languages for real-time systems, which were available at that time.

Case studies showed that time *intervals* are important in models of real-time systems. Several formalism did support modelling with intervals. But the case studies also showed the need to express the *accumulated present time* of a certain phenomenon, or state, of the systems. This accumulated present time, also called the *duration of the state*, could not be expressed by the available formalism.

This led to the introduction of DC [72] as an extension of the *Interval Temporal Logic* (ITL) of Halpern, Manna, and Moszkowski [23, 46], with the difference that DC is based on intervals of real numbers, whereas ITL is based on a discrete-time domain. The reason for basing DC on a continuous-time domain is that many of the considered applications were in the area of hybrid systems where a discrete computer component interacts with a continuous environment using sensors and actuators.

1.2 Motivating Examples

We will introduce the notion of *duration* using two simple examples as in [70]. The first example comes from the Gas Burner case study [63, 58].

A simple gas burner system: Consider a simple model of a gas burner where we observe three aspects over time. The aspects are:

- the *gas is flowing*,
- the *flame is burning*, and
- there is a *gas leak*,

and they can be modelled by two *state variables*:

$$\mathrm{Gas}, \mathrm{Flame} : \mathbb{T}\mathrm{ime} \to \{0, 1\} \ .$$

We shall use real numbers as the *time domain*, i.e.

$$\mathbb{T}\mathrm{ime} \mathrel{\widehat{=}} \mathbb{R} \ ,$$

and the intuition is that $\mathrm{Gas}(t) = 1$ if and only if (abbreviated iff), gas is leaking at time t. The state variable Flame has a similar interpretation. Furthermore, the aspect that gas leaks ($\mathrm{Leak} : \mathbb{T}\mathrm{ime} \to \{0, 1\}$) can be expressed by a Boolean combination of the two state variables above:

$$\mathrm{Leak}(t) \mathrel{\widehat{=}} \mathrm{Gas}(t) \wedge \neg \mathrm{Flame}(t) \ .$$

A Boolean combination of state variables is called a *state expression*, and it describes an aspect of a combined state in the system.

A major requirement for a gas burner system is that the amount of gas leaking should not be too much. Leaking gas cannot be prevented because gas

must be flowing a little while before it can be ignited. For a given time interval $[a, b]$, the integral

$$\int_a^b \text{Leak}(t)dt$$

is the total time the system is leaking in the interval $[a, b]$, also called the *duration* of Leak in $[a, b]$.

We shall use $\mathbb{I}\text{ntv}$ to denote the set of all time intervals:

$$\mathbb{I}\text{ntv} \mathrel{\widehat{=}} \{\, [a, b] \subseteq \mathbb{R} |\ a \leq b \} \ .$$

Scheduling of processes sharing a single processor: In connection with DC, scheduling and shared processors have been studied in several papers, e.g. [69, 67, 11]. Consider a shared processor, where n processes $\{p_1, \ldots, p_n\}$ share a single processor. To formalize the behavior of this processor, the model must capture, at least

- which processes are ready to run, and
- which process (if any) is currently running.

There are many ways in which to choose the state variables and the model below is based on [69]. For each process $p_i, 1 \leq i \leq n$, two state variables are used:

$$\begin{aligned} \text{Rdy}_i &: \mathbb{T}\text{ime} \to \{0, 1\} \\ \text{Run}_i &: \mathbb{T}\text{ime} \to \{0, 1\} \ , \end{aligned}$$

where $\text{Rdy}_i(t) = 1$ iff process p_i is ready at time t, and $\text{Run}_i(t) = 1$ iff process p_i is running at time t.

Since only ready processes may run and at most one process may run at a given time, the state variables must satisfy the well-formedness constraints:

$$\begin{aligned} \text{Run}_i(t) &\ \Rightarrow\ \text{Rdy}_i(t) \\ \text{Run}_i(t) &\ \Rightarrow\ \textstyle\bigwedge_{j \neq i} \neg \text{Run}_j(t) \ . \end{aligned}$$

Suppose that each process p_i on regular basis or demand should complete a task, and to do so it needs a certain amount $k_i \in \mathbb{R}_+$ of processing time. If p_i starts on a task at time b and finishes that task at time e, then we have that

$$\int_b^e \text{Run}_i(t)dt = k_i \ .$$

Hence, the duration of Run_i must be k_i in the interval $[b, e]$.

1.3 Informal Introduction to Duration Calculus

Duration Calculus is an example of a *modal logic* [7], where the *possible worlds* are time intervals. A consequence of this is that formulas can be considered

truth-valued functions on intervals and properties of intervals can be expressed without mentioning the intervals explicitly. An example of a formula is

$$20 \cdot \int \text{Leak} \leq \ell \ ,$$

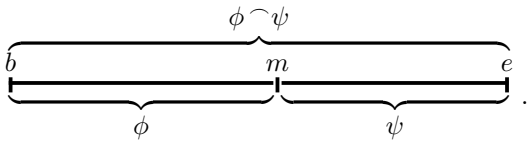
where ℓ is a special symbol denoting the length $b - a$ of the actual interval, say $[a, b]$. The formula is true on the interval $[a, b]$, iff

$$20 \int_a^b \text{Leak}(t)dt \leq b - a \ .$$

Thus, the duration of leak should at most be a twentieth of the elapsed time.

In general, atomic formulas are constructed from constants, durations and ℓ using functions and relation of real arithmetic.

Atomic formulas are combined using connectives and quantifiers of predicate logic. Furthermore, special interval *modalities* can be used. An example is the *chop modality* (written "$\frown$") from ITL: The formula $\phi \frown \psi$ (reads "ϕ chop ψ") holds on $[b, e]$, iff there exists m, where $b \leq m \leq e$, such that ϕ holds on $[b, m]$ and ψ holds on $[m, e]$:



The chop modality is an example of a binary modality. Other modalities can be derived from chop using propositional logic, for example, the unary ("for some subinterval") modality $\Diamond$, and the dual modality $\Box$ ("for all subintervals"):

$$\begin{array}{lcll} \Diamond\phi & \widehat{=} & \text{true} \frown (\phi \frown \text{true}) & \text{reads: "for some subinterval: } \phi\text{"} \\ \Box\phi & \widehat{=} & \neg\Diamond\neg\phi & \text{reads: "for all subintervals: } \phi\text{"}. \end{array}$$

Furthermore, we shall use the abbreviations:

$$\begin{array}{l} \lceil\!\lceil S \rceil\!\rceil \ \widehat{=} \int S = \ell \wedge \ell > 0 \\ \lceil\!\lceil \ \rceil\!\rceil \ \ \widehat{=} \ell = 0 \ . \end{array}$$

The formula $\lceil\!\lceil S \rceil\!\rceil$ holds for non-point intervals where the state expression S holds (has value 1) throughout the interval except for isolated points, and $\lceil\!\lceil \ \rceil\!\rceil$ holds for point intervals.

Using these abbreviations on can express a decision for design of gas burners

$$\Box(\lceil\!\lceil \text{Leak} \rceil\!\rceil \Rightarrow \ell \leq 1) \ ,$$

where gas is leaking for at most one time unit in every subinterval.

2 Syntax, Semantics and Proof System

This section will cover syntax, semantics and a proof system for Duration Calculus. The presentation is based on [70, 25], but here we will be far less detailed.

2.1 Syntax

Duration Calculus was introduced as an extension of predicate modal logic, where the first-order part is based on real arithmetic. The first-order variables are called *global variables*, and we assume that an infinite set $GVar$ of global variables ranged over by $x, y, z, \ldots$ is given. In general, we assume that there is an infinite set $FSymb$ of *global function symbols* $f^n, g^m, \ldots$ equipped with arities $n, m \geq 0$. 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, $\underline{f}^n : \mathbb{R}^n \to \mathbb{R}$. The meaning of a constant f^0 is a real number $\underline{f}^0 \in \mathbb{R}$.

Similarly, we assume we that there is an infinite set $RSymb$ of *global relation symbols* $G^n, H^m, \ldots$ equipped with arities $n, m \geq 0$. The meaning of a global relation symbol G^n, $n > 0$, will be an n-ary truth-valued function, $\underline{G}^n : \mathbb{R}^n \to \{\mathrm{tt}, \mathrm{ff}\}$. The constants true and false are the only two global relation symbols with arity 0, and the meaning is the usual one: $\underline{\mathrm{true}} = \mathrm{tt}$ and $\underline{\mathrm{false}} = \mathrm{ff}$.

When function symbols, e.g. $+$ and $-$, and relation symbols, e.g. $\geq$ and $=$, occur in formulas they appear in the usual notation and are assumed to have their standard meaning.

We have the following syntactical categories for the time-dependent part:

- An infinite set $SVar$ of *state variables* $P, Q, R, \ldots$. A state variable denotes a Boolean-valued function of time.
- A special symbol ℓ denoting the interval length.
- An infinite set $PLetter$ of *temporal propositional letters* $X, Y, \ldots$. A temporal propositional letter denotes a truth-valued interval function.

The syntactical categories for *state expressions* ($S, S_i \in SExp$), *terms* ($\theta, \theta_i \in Term$), and *formulas* ($\phi, \psi \in Formula$), are defined by the abstract syntax:

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

$$\theta ::= x \mid \ell \mid \textstyle\int S \mid f^n(\theta_1, \ldots, \theta_n)$$

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

In state expressions and formulas we shall use derived propositional connectives for conjunction $\wedge$, implication $\Rightarrow$, biimplication $\Leftrightarrow$, and standard abbreviations concerning quantifiers will be used.

Moreover, whenever $\neg, (\exists x), (\forall x), \Box$ and $\Diamond$ occur in formulas they have higher *precedence* than the binary connectives and the binary modalities $\frown$ and $\smile$ (defined below). The formula $(\Box\phi) \Rightarrow (((\forall x)(\neg\psi)) \frown \varphi)$, for example, can be written as $\Box\phi \Rightarrow ((\forall x)\neg\psi \frown \varphi)$. Furthermore, we will use standard abbreviation in connection with quantification, for example,

$$\exists x > \theta . \phi \mathrel{\widehat{=}} (\exists x)(x > \theta \wedge \phi) \; .$$

2.2 Semantics

In the semantics we shall shall assume fixed, standard interpretations of function and relation symbols of real arithmetic. The meaning of global variables is given by a *value assignment*, which is a function

$$\mathcal{V} : GVar \to \mathbb{R} ,$$

associating a real number with each global variable. Let Val be the set of all value assignments:

$$Val \;\widehat{=}\; GVar \to \mathbb{R} .$$

Two value assignments $\mathcal{V}, \mathcal{V}' \in Val$ are called *x-equivalent* if they agree on all global variables except x, i.e. if $\mathcal{V}(y) = \mathcal{V}'(y)$ for every global variable $y \neq x$.

An *interpretation* for state variables and propositional letters is a function:

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

where

- $\mathcal{I}(P) : \mathbb{T}\text{ime} \to \{0,1\}$, for every state variable P,
- $\mathcal{I}(P)$ has a finite number of discontinuity points in every interval, and
- $\mathcal{I}(X) : \mathbb{I}\text{ntv} \to \{\text{tt},\text{ff}\}$, for every propositional letter X.

Thus, each function $\mathcal{I}(P)$ has the property of *finite variability*, and, hence, $\mathcal{I}(P)$ is integrable in every interval.

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

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

defined inductively on the structure of state expressions by:

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

The function $\mathcal{I}[\![S]\!]$ has a finite number of discontinuity points in any interval and is thus integrable in every interval. In the following we will use the abbreviations: $S_{\mathcal{I}} \;\widehat{=}\; \mathcal{I}[\![S]\!]$ and $X_{\mathcal{I}} \;\widehat{=}\; \mathcal{I}(X)$.

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

$$\mathcal{I}[\![\theta]\!] : (Val \times \mathbb{I}\text{ntv}) \to \mathbb{R} ,$$

defined inductively on the structure of terms by:

$$\begin{aligned}
&\mathcal{I}[\![x]\!](\mathcal{V},[b,e]) && = \mathcal{V}(x)\\
&\mathcal{I}[\![\textstyle\int S]\!][b,e] && = \textstyle\int_b^e S_{\mathcal{I}}(t)dt\\
&\mathcal{I}[\![\ell]\!]\ (\mathcal{V},[b,e]) && = e-b\\
&\mathcal{I}[\![f^n(\theta_1,\ldots,\theta_n)]\!]\ (\mathcal{V},[b,e]) && = \underline{f}^n(c_1,\ldots,c_n)\\
& && \quad \text{where } c_i = \mathcal{I}[\![\theta_i]\!]\ (\mathcal{V},[b,e]),\ \text{for } 1\le i\le n.
\end{aligned}$$

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

$$\mathcal{I}[\![\phi]\!] : (Val \times \mathbb{I}\mathrm{ntv}) \to \{\mathrm{tt},\mathrm{ff}\}\ ,$$

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

$$\begin{aligned}
\mathcal{I},\mathcal{V},[b,e] &\models \phi \ \ \widehat{=}\ \ \mathcal{I}[\![\phi]\!]\ (\mathcal{V},[b,e]) = \mathrm{tt}\\
\mathcal{I},\mathcal{V},[b,e] &\not\models \phi \ \ \widehat{=}\ \ \mathcal{I}[\![\phi]\!]\ (\mathcal{V},[b,e]) = \mathrm{ff}\ .
\end{aligned}$$

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

- $\mathcal{I},\mathcal{V},[b,e] \models X$ iff $X_{\mathcal{I}}([b,e]) = \mathrm{tt}$.
- $\mathcal{I},\mathcal{V},[b,e] \models G^n(\theta_1,\ldots,\theta_n)$ iff $\underline{G}^n(c_1,\ldots,c_n) = \mathrm{tt}$, where $c_i = \mathcal{I}[\![\theta_i]\!](\mathcal{V},[b,e])$ for $1\le i\le n$.
- $\mathcal{I},\mathcal{V},[b,e] \models \neg\phi$ iff $\mathcal{I},\mathcal{V},[b,e] \not\models \phi$.
- $\mathcal{I},\mathcal{V},[b,e] \models \phi \vee \psi$ iff $\mathcal{I},\mathcal{V},[b,e] \models \phi$ or $\mathcal{I},\mathcal{V},[b,e] \models \psi$.
- $\mathcal{I},\mathcal{V},[b,e] \models \phi \frown \psi$ iff $\mathcal{I},\mathcal{V},[b,m] \models \phi$ and $\mathcal{I},\mathcal{V},[m,e] \models \psi$, for some $m \in [b,e]$.
- $\mathcal{I},\mathcal{V},[b,e] \models (\exists x)\phi$ iff $\mathcal{I},\mathcal{V}',[b,e] \models \phi$, for some $\mathcal{V}'$ x-equivalent to $\mathcal{V}$.

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

Examples: The validity of the following two formulas relies on the finite variability of states (Why?).

$$\lceil\!\rceil \vee (\mathrm{true} \frown \lceil S \rceil) \vee (\mathrm{true} \frown \lceil \neg S \rceil) \tag{1}$$

$$\lceil\!\rceil \vee (\lceil S \rceil \frown \mathrm{true}) \vee (\lceil \neg S \rceil \frown \mathrm{true})\ . \tag{2}$$

The next three formulas express basic properties of durations:

$$\textstyle\int S + \int \neg S = \ell\ , \qquad \int S \le \ell \qquad \text{and} \qquad \int S_1 \ge \int S_2,\ \text{if } S_2 \Rightarrow S_1,$$

and the following formulas are valid formulas about $\lceil S \rceil$:

$$\lceil S \rceil \Leftrightarrow (\lceil S \rceil \frown \lceil S \rceil) \qquad \text{and} \qquad (\lceil S_1 \rceil \wedge \lceil S_2 \rceil) \ \Leftrightarrow\ \lceil S_1 \wedge S_2 \rceil\ ,$$

where the first formula holds because we have a continuous time domain, and the last formula reflects the structure of state expressions.

2.3 Proof System

The proof system has two parts: The first part is a based on the proof system S' for IL presented and shown complete wrt. *abstract* value and time domains in [13]. The second part is proof system for state durations, which is shown relative complete wrt. IL in [27].

Proof system for interval logic: The proof system S' is a Hilbert style proof system, where we shall use the following notions. *Free (global) variables* as known from predicate logic. Furthermore, a term is called *flexible* if ℓ or a duration $\int S$ occur in the terms, and a formula is called *flexible* if ℓ, a duration $\int S$ or a propositional letter occur in the formula. A term or formula which is not flexible is also called *rigid*. Note that a rigid formula may contain the chop modality.

Furthermore, we shall use the *dual-chop* modality $\phi \mathbin{⌣} \psi$ defined by:

$$\phi \mathbin{⌣} \phi \mathrel{\widehat{=}} \neg((\neg\phi) \frown (\neg\psi)) \qquad \text{reads: “}\phi \text{ dual-chop } \psi\text{”} .$$

The reading of $\phi \mathbin{⌣} \psi$ is as follows: $\phi \mathbin{⌣} \psi$ holds on $[b, e]$ iff, for all $m \in [b, e]$: ϕ holds on $[b, m]$ or ψ holds on $[m, e]$.

The axioms and rules below are basically those of [13], except that we use the abbreviation for the dual modality of chop. This has the advantage that certain axioms and rules can be expressed more succinctly (avoiding double negation), and some proof have a more compact form.

The axioms of are:

A0 $\quad \ell \geq 0 .$

A1
$$((\phi \frown \psi) \wedge (\neg\phi \mathbin{⌣} \varphi)) \;\Rightarrow\; (\phi \frown (\psi \wedge \varphi)) .$$
$$((\phi \frown \psi) \wedge (\varphi \mathbin{⌣} \neg\psi)) \;\Rightarrow\; ((\phi \wedge \varphi) \frown \psi) .$$

A2 $\quad ((\phi \frown \psi) \frown \varphi) \;\Leftrightarrow\; (\phi \frown (\psi \frown \varphi)) .$

R
$$(\phi \frown \psi) \Rightarrow \; \phi \;\text{ if } \phi \text{ is a rigid formula.}$$
$$(\phi \frown \psi) \Rightarrow \; \psi \;\text{ if } \psi \text{ is a rigid formula.}$$

E
$$(\exists x.\phi \frown \psi) \Rightarrow \; \exists x.(\phi \frown \psi) \;\text{ if } x \text{ is not free in } \psi.$$
$$(\phi \frown \exists x.\psi) \Rightarrow \; \exists x.(\phi \frown \psi) \;\text{ if } x \text{ is not free in } \phi.$$

L1
$$((\ell = x) \frown \phi) \Rightarrow \; ((\ell \neq x) \mathbin{⌣} \phi) .$$
$$(\phi \frown (\ell = x)) \Rightarrow \; (\phi \mathbin{⌣} (\ell \neq x)) .$$

L2 $\quad (x \geq 0 \wedge y \geq 0) \Rightarrow \; ((\ell = x + y) \Leftrightarrow \; ((\ell = x) \frown (\ell = y))) .$

L3
$$\phi \Rightarrow \; (\phi \frown (\ell = 0))$$
$$\phi \Rightarrow \; ((\ell = 0) \frown \phi) .$$

The inference rules of are:

MP	if ϕ and $\phi \;\Rightarrow\; \psi$ then ψ.	(modus ponens)
G	if ϕ then $(\forall x)\phi$.	(generalisation)
N	if ϕ then $\phi \frown$ false. if ϕ then false $\frown \phi$.	(necessity)
M	if $\phi \;\Rightarrow\; \psi$ then $(\phi \frown \varphi) \;\Rightarrow\; (\psi \frown \varphi)$. if $\phi \;\Rightarrow\; \psi$ then $(\varphi \frown \phi) \;\Rightarrow\; (\varphi \frown \psi)$.	(monotonicity)

Predicate logic: The proof system also contains axioms of first order predicate logic with equality. Extra care must be taken when universally quantified formulas are instantiated and when an existential quantifier is introduced as shown below.

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. Furthermore, a formula is called *chop free* if $\frown$ does not occur in the formula.

Two axiom schemes for quantification are:

$$\begin{array}{ll} \text{Q1} & \forall x.\phi(x) \;\Rightarrow\; \phi(\theta) \\ \text{Q2} & \phi(\theta) \;\Rightarrow\; \exists x.\phi(x) \end{array} \qquad \left(\begin{array}{l} \text{if } \theta \text{ is free for } x \text{ in } \phi(x), \text{ and} \\ \text{either } \theta \text{ is rigid or } \phi(x) \text{ is chop free} \end{array}\right) .$$

The proof system has to contain axioms for first order logic of real arithmetic. We will not be explicit about other axioms and rules of real arithmetic. We just write PL in proofs, when we exploit properties of real arithmetic.

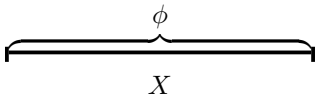
Proof system for state durations: The axioms and rules for DC must reflect the structure of state expressions. The axioms are:

DCA1	$\int 0 = 0$.
DCA2	$\int 1 = \ell$.
DCA3	$\int S \geq 0$.
DCA4	$\int S_1 + \int S_2 = \int(S_1 \vee S_2) + \int(S_1 \wedge S_2)$.
DCA5	$((\int S = x) \frown (\int S = y)) \Rightarrow (\int S = x + y)$.
DCA6	$\int S_1 = \int S_2$, provided $S_1 \Leftrightarrow S_2$ holds in propositional logic.

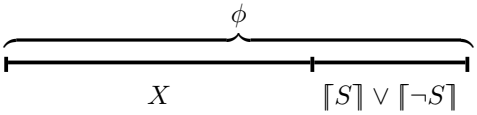
We also need rules to formalize the finite variability of state expressions. To this end, the notion *state induction* is introduced. The main idea of state induction is the following. To prove that ϕ holds for every interval, it suffices to establish:

– The base case: ϕ holds for point intervals.

– The inductive step: it is established that ϕ holds for an interval of the form $X \frown (\lceil\!\lceil S \rceil\!\rceil \vee \lceil\!\lceil \neg S \rceil\!\rceil)$, under the assumption that $X \Rightarrow \phi$. Hence, from a arbitrary interval X on which ϕ holds:



we can conclude that ϕ holds for a larger interval, where X is extended by a section throughout which either S or $\neg S$ hold:



These two steps suffices since every interval can be covered by a finite sequence of sections for which either $\lceil\!\lceil S \rceil\!\rceil$ or $\lceil\!\lceil \neg S \rceil\!\rceil$ holds.

Let $H(X)$ be a formula containing the propositional letter X and let $S_1, \ldots, S_n$ be any finite collection of state expressions which are *complete* in the sense that

$$(\bigvee_{i=1}^{n} S_i) \;\Leftrightarrow\; 1 \; .$$

For a complete collection of state expressions: $S_1, \ldots, S_n$, there are two *induction rules*:

IR1 If $H(\lceil\!\lceil\, \rceil\!\rceil)$ and $H(X) \Rightarrow H(X \vee \bigvee_{i=1}^{n}(X \frown \lceil\!\lceil S_i \rceil\!\rceil))$ then $H(\text{true})$

and

IR2 If $H(\lceil\!\lceil\, \rceil\!\rceil)$ and $H(X) \Rightarrow H(X \vee \bigvee_{i=1}^{n}(\lceil\!\lceil S_i \rceil\!\rceil \frown X))$ then $H(\text{true})$,

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

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

Deduction and proof: A *deduction of ϕ from a set of formulas Γ* is a sequence of formulas

$$\begin{array}{c} \phi_1 \\ \vdots \\ \phi_n \; , \end{array}$$

where ϕ_n is ϕ, and each ϕ_i is either a member of Γ, an instance of one of the above axiom schemes or obtained by applying one of the above inference rules to previous members of the sequence. We write $\Gamma \vdash \phi$ to denote that there exists a

deduction of ϕ from Γ, and we write $\Gamma, \phi \vdash \psi$ for $(\Gamma \cup \{\phi\}) \vdash \psi$. When $\Gamma = \emptyset$, the deduction is called a *proof* of ϕ. In this case we call ϕ a *theorem* $\vdash \phi$.

As an example, we derive the monotonicity rules for the dual of chop:

IL1 $\quad \phi \Rightarrow \psi \vdash (\phi \chopd \varphi) \Rightarrow (\psi \chopd \varphi)$
$\quad\quad\;\; \phi \Rightarrow \psi \vdash (\varphi \chopd \phi) \Rightarrow (\varphi \chopd \psi)$.

Proof. Here is a deduction establishing the first part:

1.	$\phi \Rightarrow \psi$	assumption
2.	$\neg\psi \Rightarrow \neg\phi$	1., PL
3.	$\neg(\psi \chopd \varphi) \Rightarrow \neg\psi \chop \neg\varphi$	def. $\chopd$, PL
4.	$(\neg\psi \chop \neg\varphi) \Rightarrow (\neg\phi \chop \neg\varphi)$	2., M
5.	$\neg(\psi \chopd \varphi) \Rightarrow (\neg\phi \chop \neg\varphi)$	4., def. $\chopd$, PL
6.	$(\phi \chopd \varphi) \Rightarrow (\psi \chopd \varphi)$	5., def. $\chopd$, PL

The following two deduction theorems can be used to simplify proofs.

Theorem 1. *[28] If a deduction*

$$\Gamma, \phi \vdash \psi$$

involves

- *no application of the generalization rule* G *in which the quantified variable is free in ϕ, and*
- *no application of the induction rules,* IR1 *and* IR2*, in which the induction letter occurs in ϕ,*

then

$$\Gamma \vdash \Box\phi \Rightarrow \psi \ .$$

Theorem 2. *[70] Suppose that $\{S_1, \ldots, S_n\}$ is a complete set of state expressions. Then*

$$\left.\begin{array}{l} \Gamma \vdash H(\llbracket\ \rrbracket) \text{ and} \\ \Gamma, H(X) \vdash H(X \vee \bigvee_{i=1}^{n}(X \chop \llbracket S_i \rrbracket)) \end{array}\right\} \text{ implies } \Gamma \vdash H(\mathit{true}) \ ,$$

and

$$\left.\begin{array}{l} \Gamma \vdash H(\llbracket\ \rrbracket) \text{ and} \\ \Gamma, H(X) \vdash H(X \vee \bigvee_{i=1}^{n}(\llbracket S_i \rrbracket \chop X)) \end{array}\right\} \text{ implies } \Gamma \vdash H(\mathit{true}) \ ,$$

provided the deductions from $\Gamma, H(X)$ involve no application of the induction rules, where the induction letter occurs in $H(X)$.

Consider the formulas:

DC1 $\quad \llbracket\ \rrbracket \vee (\mathrm{true} \chop \llbracket S \rrbracket) \vee (\mathrm{true} \chop \llbracket \neg S \rrbracket)$

DC2 $\quad \llbracket\ \rrbracket \vee (\llbracket S \rrbracket \chop \mathrm{true}) \vee (\llbracket \neg S \rrbracket \chop \mathrm{true})$.

The proof of DC1, for example, is by induction using $H(X) \;\widehat{=}\; X \Rightarrow \mathrm{DC1}$ as induction hypothesis and $\{S, \neg S\}$ as a complete collection of state expressions. Using Theorem 2 (and propositional logic), the proof is completed by establishing the base case $H(\lceil\rceil)$, i.e. $\lceil\rceil \Rightarrow \mathrm{DC1}$, which is trivial, and three easy deductions:

- $(X \Rightarrow \mathrm{DC1}) \;\vdash\; X \Rightarrow \mathrm{DC1}$,
- $(X \Rightarrow \mathrm{DC1}) \;\vdash\; (X \frown \lceil S \rceil) \Rightarrow \mathrm{DC1}$, and
- $(X \Rightarrow \mathrm{DC1}) \;\vdash\; (X \frown \lceil \neg S \rceil) \Rightarrow \mathrm{DC1}$.

An essential property of a proof system is that every theorem is valid.

Theorem 3. *(Soundness)*

$$\vdash \phi \textit{ implies } \models \phi\ .$$

The soundness of axioms and inference rules of IL are treated in [13] and axioms for DC are simple. The soundness of IR1 and IR2 relies on the finite variability of states and we refer to [70] for a proof.

Another important property of a proof system is that it is complete, i.e. every valid formula is provable. As DC extends real number arithmetic, and, furthermore, natural number reasoning are used in several case studies, the completeness issue is a complex matter.

We will not go into details concerning completeness issues for the arithmetical parts, but mention a few classical results. In 1951, Tarski [64] proved a completeness and decidability results for a theory of reals, where atomic formulas involve equality ($=$) and ordering relations ($<, \leq, >, \geq$) and terms are constructed from rational constants and (global) variables using operations for addition, subtraction, negation, and multiplication. For natural number theory, Presburger gave in 1930 a decision algorithm for linear arithmetic (excluding multiplication), while Gödel, in 1931, established his famous incompleteness theorem for a theory having addition, subtraction, negation, and multiplication as operations. Concerning theorem proving with real numbers, we refer to [29].

For DC, there is a relative completeness result with respect to ITL [27, 70], which shows that there is a deduction for every valid formula, from the collection of valid ITL formulas. The completeness of IL, with abstract value and time domains, is proved in [12], and in [21] there is a completeness result for DC with respect to abstract value and time domains.

3 Basic Decidability and Undecidability Results

It is a very tedious and error-prone task to write proof by hand using a formal system. DC proofs are no exception. So there is a natural desire to get tool support. Interval logics are, however, typically very expressive logics, which often are undecidable. For example, the propositional interval logic HS with unary modalities *begins, ends* and their inverses, by Halpern and Shoham [24], is shown highly undecidable for a collection of classes of interval models. In this section we will review the first results on decidable as well as undecidable fragments of DC [71], to show limits of what we can hope for. In Sect. 5 and Sect. 6 we will provide some extensions to these results.

3.1 A Basic Decidability Results

Consider first a simple subset of DC called *Restricted Duration Calculus* (*RDC*). The formulas of *RDC* are constructed by the following abstract syntax:

$$S ::= 0 \mid 1 \mid P \mid \neg S_1 \mid S_1 \vee S_2$$
$$\phi ::= \lceil\!\lceil S \rceil\!\rceil \mid \neg\phi \mid \phi \vee \psi \mid \phi \frown \psi \,.$$

Decidability results were established both for discrete and continuous time interpretations. In a *discrete time interpretation* state variables are allowed to change value at natural number time points only. Furthermore, only intervals having natural number end points are considered, and, at last, chop points must be natural numbers.

Theorem 4. *[71] The satisfiability problem for RDC is decidable for discrete and continuous time.*

The theorem is proved by reducing the satisfiability problem for *RDC* to the emptiness problem for regular languages, which is decidable. The main idea of this reduction is that a letter in the alphabet describes a section of an interpretation. A letter is a conjunction of state variables or negation of state variables. $\lceil\!\lceil S \rceil\!\rceil$ is translated L^+, where L is the set of letters "for which S is true". Disjunction, negation and chop correspond to union, complement and concatenation, respectively, of regular languages.

The complexity of the satisfiability problem for *RDC* is non-elementary. Peter Sestoft established this result for discrete time and for continuous time the result is shown in [52].

In the tool DCVALID [50] an extension of discrete-time *RDC* with quantification of state variables is translated into Monadic Second-Order Logic over finite strings, which is a slight variant of the Weak Monadic Second-Order theory of one successor (WS1S) [10, 14]. This second-order theory is decidable and used for instance in the MONA system [40].

There are certainly more results than those mentioned above. For example, decidable subsets are also considered in [26, 42, 49, 17, 59, 36, 16]. References [61, 51] concern implementation of tools to check the validity of a subclass of Duration Calculus and its higher-order version. In [18], there is a bounded model construction for discrete-time Duration Calculus, which is shown NP-complete. Furthermore, in [15], a robust interpretation for a subset of Duration Calculus is considered, and a semi-decision result is obtained. Model-checking formulas wrt. automata based implementations is considered in [39, 73, 43, 44, 9, 38, 16], and automated proof assistance is considered in [62, 60, 45, 56, 55, 54].

3.2 Basic Undecidability Results

From a point of view of tool support, a disappointing fact is that seemingly small extensions to *RDC* are shown undecidable in [71] by reducing the halting problem of two-counter machines to the satisfiability problem. The subsets considered were:

– RDC_1, which is defined by:

$$\phi ::= \ell = 1 \mid \lceil S \rceil \mid \neg\phi \mid \phi \vee \psi \mid \phi \frown \psi \ .$$

– RDC_2, which is defined by:

$$\phi ::= \int S_1 = \int S_2 \mid \neg\phi \mid \phi \vee \psi \mid \phi \frown \psi \ .$$

– RDC_3, which is defined by:

$$\phi ::= \ell = x \mid \lceil S \rceil \mid \neg\phi \mid \phi \vee \psi \mid \phi \frown \psi \mid \forall x.\phi \ .$$

The satisfiability problem for RDC_1 is decidable for a discrete time interpretation as the formula $\ell = 1$ is expressible in RDC as $\lceil 1 \rceil \wedge \neg(\lceil 1 \rceil \frown \lceil 1 \rceil)$. But for a continuous time domain, the satisfiability problem for RDC_1 is undecidable.

Theorem 5. *[71] The satisfiability problem for RDC_1 is undecidable for continuous time.*

The main idea behind the proof of this theorem is to reduce the (undecidable) halting problem for 2-counter machines to the satisfiability problem of RDC_1. In this case, a value m of a counter is represented by a state variable C which has m sections of the form $\lceil C \rceil$ separated by sections of the form $\lceil \neg C \rceil$. The continuous time domain and the formula $\ell = 1$ are used to represent any counter value on a unit interval. A full configuration of a 2-counter machine can be represented on an interval of length 4 (two units for the counters, one for the current state, and one is used as a separator). Furthermore, the formulas can express transitions of the machine from one configuration to the next.

Theorem 6. *[71] The satisfiability problems for RDC_2 and RDC_3 are undecidable for discrete and continuous time.*

The main idea behind the encoding of a 2-counter machine in RDC_2 is to have two state variables C^+ and C^- for a counter c, so that the value of c is $\int C^+ - \int C^-$ interpreted over the interval representing the computation up to the current state. The value of c is increased (decreased) by one by letting $\lceil C^+ \rceil$ ($\lceil C^- \rceil$) hold over the next section. To simulate the 2-counter machine we just need the formula $\int C^+ = \int C^-$ to test whether c's value is 0, but it is not necessary to compute the actual value of the counter. The sections representing counter operations should have equal length and RDC_2 is strong enough to express that. The proof idea works for discrete as well as a continuous time domain. We will not go into details about RDC_3, as this undecidability result just shows the well-known power of first-order logic. For further details, we refer to [71, 70].

The non-elementary complexity bound of the satisfiability problem for RDC shows that very "heavy" tools are necessary to handle this seemingly simple subset. Furthermore, the simple extension with $\ell = 1$ (to RDC_2) leads to undecidability (wrt. continuus time), and so does the extension with $\int S_1 = \int S_2$ leading to RDC_3. Hence, it is not easy to find decidable extensions to RDC where you can express precise time bounds like $\ell = 1$, or make assertion about the duration of states. Before we extend the decidability results, we first show that RDC can be simplified [8].

3.3 *RDC**: A Simplification of *RDC*

The simplified fragment, called RDC^*, has the following abstract syntax:

$$\phi ::= P \mid \pi \mid \neg\phi \mid \phi \vee \psi \mid \phi \frown \psi \ ,$$

where P is a state variable. In RDC^*, π stands for point interval and P corresponds to the formula $\lceil P \rceil$ of RDC. The simplification is that RDC^* has no syntactical category for state expressions.

The interesting point is that RDC^* has the same expressive power as RDC, as shown by the translation $\cdot^*$ from the formulas of RDC into formulas of RDC^*:

$$\begin{array}{lll} \lceil 0 \rceil^* & = \neg \mathrm{T} & \lceil 1 \rceil^* = \neg\pi \\ \lceil P \rceil^* & = P & \lceil \neg S \rceil^* = \neg\pi \wedge \Box\neg(\lceil S \rceil^*) \\ \lceil S_1 \vee S_2 \rceil^* & = \neg\pi \wedge \Box(\neg\pi \rightarrow \Diamond(\lceil S_1 \rceil^* \vee \lceil S_2 \rceil^*)) & (\neg\phi)^* = \neg(\phi^*) \\ (\phi \vee \psi)^* & = \phi^* \vee \psi^* & (\phi \frown \psi)^* = \phi^* \frown \psi^* \ . \end{array}$$

This translation shows that Boolean connectives in state expressions "behave like modalities. The correctness of the translation is stated in the following lemma, which says that truth is preserved by the translation $\cdot^*$. The lemma also establishes that the simpler language RDC* has the same expressive power as RDC.

Lemma 1. *[8] For all RDC formulas ϕ, interpretations $\mathcal{I}$, and intervals $[t, u]$:*

$$\mathcal{I}, [t, u] \models \phi^* \quad \textit{iff} \quad \mathcal{I}, [t, u] \models \phi.$$

Discrete time: Lemma 1 holds for discrete time as well. Even in a seemingly simple fragment, some real-time properties are expressible. Observe first that the formula $\ell = 1$ can be represented in RDC as follows:

$$\ell = 1 \ \widehat{=} \ \lceil 1 \rceil \wedge \neg(\lceil 1 \rceil \frown \lceil 1 \rceil) \ .$$

Simple notions of durations are also expressible in RDC and hence also in RDC^*. For example, the formula $\int S = 1$ is expressed as

$$(\lceil \, \rceil \vee \lceil \neg S \rceil) \frown (\lceil S \rceil \wedge \ell = 1) \frown (\lceil \, \rceil \vee \lceil \neg S \rceil) \ .$$

The above two formulas are not expressible in continuous-time RDC (see [71, 70]). For further discussion about the expressibility of RDC we refer to [70].

4 Duration Calculus with Iterations

In this section, we present an extension of Duration Calculus, called DC*, with the modality *iteration*, also known as *Kleene star*. Iteration was introduced to DC to facilitate the reasoning about repetitive behaviour. Iteration is particularly important for the description of the repetitive behaviour of timed automata in DC. In [37] we developed a method for designing real-time hybrid systems from

specifications written using a subset of DC* which consists of the so-called *simple* DC* formulas. One can reason about the correctness of designs in terms of the semantics of DC*. However, it would be more practical and interesting to be able to prove correctness syntactically. This requires the development of a proof system for DC*.

Let us return to the Gas Burner example presented earlier. The DC formula

$$S \rightleftharpoons \Box(\ell > 60 \Rightarrow (20 \int \mathit{Leak} \leq \ell)) \tag{3}$$

specifies that a gas burner can be in the *Leak* state for no more than one-twentieth of the time in any time interval that is at least 1 minute long. Consider the gas burner design described by the real-time automaton shown in Fig. 1. We assume that *Leak* is the initial state for the sake of simplicity. In this design *Leak* becomes detected within 1 second and leaks are separated by at least 30 seconds. This can be specified by the DC* formula

$$D \rightleftharpoons ((\lceil\!\lceil \mathit{Leak} \rceil\!\rceil \wedge \ell \leq 1)^\frown(\lceil\!\lceil \mathit{Nonleak} \rceil\!\rceil \wedge \ell \geq 30))^* . \tag{4}$$

To show the correctness of the design, we have to prove $D \Rightarrow S$. This can be done by means of our axioms about iteration given in this section.

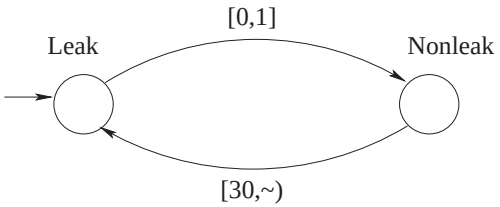


Fig. 1. A Simple Gas Burner Design

In this section we study the deductive power of three groups of axioms and a rule for iteration in DC which we add to the proof system presented in Sect. 2.

In the sequel, the set of the variables which have free occurrences in a formula φ is denoted by $FV(\varphi)$. For sets of formulas Γ, $FV(\Gamma)$ is defined as $\bigcup_{\varphi\in\Gamma} FV(\varphi)$. The state variables occurring freely a formula φ are assumed to be in $FV(\varphi)$ too.

4.1 Iteration Formulas

DC is extended by iteration by allowing formulas of the form φ^*. The semantics of formulas of this form is defined as:

$\mathcal{I}, \mathcal{V}, [b, e] \models \varphi^*$ iff either $b = e$, or there exist $m_0, m_1, \ldots, m_m$ such that
$b = m_0 < m_1 < \ldots < m_n = e$ and
$\mathcal{I}, \mathcal{V}, [m_i, m_{i+1}] \models \varphi$ for $i = 0, \ldots, n-1$.

Iteration * binds more tightly than $\frown$ and the propositional connectives.

4.2 Axioms and a Rule about Iteration in DC

From the definition of the semantics of iteration, it is obvious that the following two axioms are needed for the DC proof system:

$$\begin{array}{ll} (\mathrm{DC}^*1) & \ell = 0 \Rightarrow \varphi^* \\ (\mathrm{DC}^*2) & (\varphi^* \frown \varphi) \Rightarrow \varphi^* \end{array}$$

However, DC^*1 and DC^*2 are not enough to characterise the definition of iteration. In the following, we develop three axioms more named DC^*3, DC^*4 and DC^*5, and show that combining any of them with DC^*1 and DC^*2 will completely characterise the meanings of iteration.

$$(\mathrm{DC}^*3)\quad ((\varphi^* \wedge \psi)\frown\mathrm{true}) \Rightarrow ((\psi \wedge \ell = 0)\frown\mathrm{true}) \vee ((((\varphi^* \wedge \neg\psi)\frown\varphi) \wedge \psi)\frown\mathrm{true})$$

To understand the meaning of DC*3, assume that some non-point initial subinterval $[b, e']$ of the reference interval $[b, e]$ satisfies ψ and φ^*, and $\mathcal{I}, \mathcal{V}, [b, b] \not\models \psi$. By the definition of the semantics of iteration, for some $m_0, \ldots, m_n$ such that $b = m_0 < m_1 < \ldots < m_n = e'$ it holds that $\mathcal{I}, \mathcal{V}, [m_i, m_{i+1}] \models \varphi$. Let $j \leq n$ be the least such that $\mathcal{I}, \mathcal{V}, [b, m_j] \models \psi$. From our assumption, j exists and $j > 0$. Hence, $\mathcal{I}, \mathcal{V}, [b, m_{j-1}] \models \varphi^* \wedge \neg\psi$, and consequently, $\mathcal{I}, \mathcal{V}, [b, m_j] \models ((\varphi^* \wedge \neg\psi)\frown\varphi) \wedge \psi$.

$$(\mathrm{DC}^*4)\qquad \Box(\ell = 0 \vee (\psi\frown\varphi) \Rightarrow \psi) \Rightarrow (\varphi^* \Rightarrow \psi),$$

DC*4 is an expression of the fact that $\tilde{I}(\varphi^*)$ is the least set of time intervals $X \subseteq \mathbf{I}(T)$ which satisfies the inclusion

$$\tilde{I}(\ell = 0) \cup \tilde{I}(\varphi)\frown X \subseteq X.$$

For simplicity, let us denote

$$\mathsf{f}(\varphi, Q) \rightleftharpoons \neg(((\mathrm{true}\frown\lceil\!\lceil Q \rceil\!\rceil) \vee \ell = 0)\frown(\lceil\!\lceil \neg Q \rceil\!\rceil \wedge \neg\varphi)\frown(\lceil\!\lceil Q \rceil\!\rceil\frown\mathrm{true}) \vee \ell = 0).$$

This formula $\mathsf{f}(\varphi, Q)$ means that every maximal non-trivial subinterval of the reference interval which satisfies $\lceil\!\lceil \neg Q \rceil\!\rceil$ satisfies φ. Let

$$\mathsf{g}(\varphi, P) \rightleftharpoons \mathsf{f}(\varphi, P) \wedge \mathsf{f}(\varphi, \neg P).$$

This formula means that all maximal non-trivial subintervals of the reference interval $[b, e]$ which satisfy either $\lceil\!\lceil P \rceil\!\rceil$ or $\lceil\!\lceil \neg P \rceil\!\rceil$ satisfy φ. Because of the finite variability of state variables in DC, these intervals form a finite partition of $[b, e]$. Therefore, if the model $(\mathcal{I}, \mathcal{V}, [b, e])$ satisfies $\mathsf{g}(\varphi, P)$, then it also satisfies φ^*. This observation is captured by the rule

$$(\mathrm{DC}^*5)\quad \frac{\Gamma \vdash (\alpha\frown\mathsf{g}(\varphi, P)\frown\beta) \Rightarrow \psi}{\Gamma \vdash (\alpha\frown\varphi^*\frown\beta) \Rightarrow \psi}, \text{ where } P \notin FV(\Gamma \cup \{\varphi, \psi, \alpha, \beta\}).$$

If we substitute $\ell = 0$ for both α and β in the rule, we get a simpler version of this rule:

$$\frac{\Gamma \vdash \mathsf{g}(\varphi, P) \Rightarrow \psi}{\Gamma \vdash \varphi^* \Rightarrow \psi}, \text{ where } P \notin FV(\Gamma \cup \{\varphi, \psi\})$$

As mentioned above, if a reference interval $[b, e]$ satisfies $\mathsf{g}(\varphi, P)$, then the time points at which P changes its value inside φ partition it into subintervals which satisfy φ, and therefore $[b, e]$ itself satisfies φ^*. Given a *concrete* interpretation of P, the points at which P changes its value define a *concrete* finite partition of $[b, e]$, whereas for $[b, e]$ to satisfy φ^* we just need the *existence* of such a partition. Therefore, the side condition $P \notin FV(\Gamma \cup \{\varphi, \psi\})$ is needed to make the existence of such a partition independent from the interpretation of the variables that contribute to the truth value of formulas in $\Gamma \cup \{\varphi, \psi\}$.

Note that the scope of the soundness of $DC^*1 - DC^*4$ is, in fact, the extension ITL^* of ITL by iteration, because these axioms involve no DC-specific constructs.

It has been shown in [19] that with the additional axioms and rule DC*1, DC*2 and DC*5 for iteration, the proof system of DC is complete in the abstract-time domain for DC^*.

4.3 Interderivability between DC*3, DC*4 and DC*5

The proof rule DC*5 is implicitly related to the state variable binding existential quantifier in DC*. The key ingredient in this rule is the formula $\mathsf{g}(\varphi, P)$. The proof that DC*5 is derivable from DC*1, DC*2 and DC*4 involved the proof system for the higher order Duration Calculus which is not presented in this chapter. Readers are referred to [19] for the proof details.

The relationship between the axioms introduced for iteration is formulated as follows.

Proposition 1. *[19]*

1. *DC*4 is provable in the extension of the proof system for DC by just DC*3.*
2. *DC*3 and DC*4 is provable in the extension of the proof system for DC by the axioms DC*1, DC*2 and DC*5.*

4.4 Examples of the Use of DC*1–DC*5

In this subsection we give derivations for a couple of DC* theorems of general interest and use one of them in a proof about our introductory gas-burner example in order to give some such illustration with a practical flavour.

Here are two derivations of the monotonicity of iteration. One of them involves DC*3:

$$\begin{aligned}
\alpha^* \wedge \neg\beta^* &\Rightarrow ((\neg\beta^* \wedge \ell = 0)^\frown\text{true}) \vee ((((\alpha^* \wedge \beta^*)^\frown\alpha) \wedge \neg\beta^*)^\frown\text{true}) && \text{by } DC^*3\\
&\Rightarrow ((((\alpha^* \wedge \beta^*)^\frown\alpha) \wedge \neg\beta^*)^\frown\text{true}) && \text{by } DC^*1\\
&\Rightarrow (\neg\beta^* \wedge \beta^*)^\frown\text{true} && \text{by } DC^*_2\\
& && \text{and } \alpha \Rightarrow \beta\\
&\Rightarrow \text{false}
\end{aligned}$$

The other involves the proof rules ω and DC*5:

$1\ \left(\begin{array}{c}\lceil\!\lceil P\rceil\!\rceil\vee\\ \lceil\!\lceil\neg P\rceil\!\rceil\end{array}\right)^k \Rightarrow \left(\mathsf{g}(\alpha,P)\Rightarrow\left(\Box(\varphi\Rightarrow\beta)\Rightarrow\bigvee\limits_{l\le k}\beta^l\right)\right)$	$k<\omega$, DC
$2\ \beta^m\Rightarrow\beta^*$	$m<\omega$, DC*1, DC*2, DC
$3\ (\lceil\!\lceil P\rceil\!\rceil\vee\lceil\!\lceil\neg P\rceil\!\rceil)^k\Rightarrow(\mathsf{g}(\alpha,P)\Rightarrow(\Box(\varphi\Rightarrow\beta)\Rightarrow\beta^*))$	$k<\omega$, 1, 2, DC
$4\ \mathsf{g}(\alpha,P)\Rightarrow(\Box(\varphi\Rightarrow\beta)\Rightarrow\beta^*)$	3, ω
$5\ \alpha^*\Rightarrow(\Box(\varphi\Rightarrow\beta)\Rightarrow\beta^*)$	4, DC*5
$6\ \Box(\varphi\Rightarrow\beta)\Rightarrow(\alpha^*\Rightarrow\beta^*)$	

Here follows another useful DC* theorem:

$$\begin{array}{l}\vdash_{\mathrm{DC}^*}\ \Box(\varphi\Rightarrow\neg(\mathrm{true}^\frown\neg\alpha)\wedge\neg(\neg\beta^\frown\mathrm{true}))\wedge\\ \qquad\Box(\ell=0\Rightarrow\alpha\wedge\beta)\wedge\Box(\beta\Rightarrow\neg(\mathrm{true}^\frown\neg\gamma))\Rightarrow\\ \qquad\Rightarrow\varphi^*\Rightarrow\Box(\gamma\vee(\alpha^\frown\varphi^*{}^\frown\beta))\,.\end{array}\qquad(5)$$

To prove it in our system, below we give a derivation of $\varphi^*\Rightarrow\Box(\gamma\vee(\alpha^\frown\varphi^*{}^\frown\beta))$ using

$$\varphi\Rightarrow\neg(\mathrm{true}^\frown\neg\alpha),\varphi\Rightarrow\neg(\neg\beta^\frown\mathrm{true}),\beta\Rightarrow\neg(\mathrm{true}^\frown\neg\gamma)\text{ and }\ell=0\Rightarrow\alpha,\ell=0\Rightarrow\beta$$

as assumptions. Then (5) will follow by the deduction theorem for DC [28].

$1\ \varphi\Rightarrow\neg(\mathrm{true}^\frown\neg\alpha)$	assumption
$2\ \varphi\Rightarrow\neg(\mathrm{true}^\frown\neg(\alpha^\frown\ell=0))$	by 1
$3\ \ell=0\Rightarrow\varphi^*$	by DC_1^*
$4\ \varphi\Rightarrow\neg(\mathrm{true}^\frown\neg(\alpha^\frown\varphi^*))$	by 2, 3, $Mono_r$
$5\ \ell=0\Rightarrow\alpha$	assumption
$6\ \ell=0\Rightarrow(\ell=0^\frown\ell=0)$	$L2$
$7\ \ell=0\Rightarrow(\alpha^\frown\varphi^*))$	by 5, 6, DC_1^*, $Mono_l$, $Mono_r$
$8\ (\mathrm{true}^\frown\neg(\alpha^\frown\varphi^*))\Rightarrow\neg\ell=0$	by 7, DC
$9\ \neg((\mathrm{true}^\frown\neg(\alpha^\frown\varphi^*))\wedge\ell=0^\frown\mathrm{true})$	by 8, N_l
$10\ (\varphi^*\wedge(\mathrm{true}^\frown\neg(\alpha^\frown\varphi^*)))\Rightarrow$ $(((\mathrm{true}^\frown\neg(\alpha^\frown\varphi^*))\wedge\ell=0)^\frown\mathrm{true})\vee$ $(((\varphi^*\wedge\neg(\mathrm{true}^\frown\neg(\alpha^\frown\varphi^*))^\frown\varphi)\wedge$ $(\mathrm{true}^\frown\neg(\alpha^\frown\varphi^*)))^\frown\mathrm{true})$	by DC^*3
$11\ (\varphi^*\wedge\neg(\mathrm{true}^\frown\neg(\alpha^\frown\varphi^*))^\frown\varphi)\wedge$ $(\mathrm{true}^\frown\neg(\alpha^\frown\varphi^*))\Rightarrow$ $(\mathrm{true}^\frown(\alpha^\frown\varphi^*{}^\frown\varphi)\wedge\neg(\alpha^\frown\varphi^*))\vee$ $(\varphi\wedge(\mathrm{true}^\frown\neg(\alpha^\frown\varphi^*))))$	DC
$12\ (\alpha^\frown\varphi^*{}^\frown\varphi)\Rightarrow(\alpha^\frown\varphi^*)$	by DC_2^*, $Mono_r$
$13\ \neg((\alpha^\frown\varphi^*{}^\frown\varphi)\wedge\neg(\alpha^\frown\varphi^*))$ $\vee(\varphi\wedge(\mathrm{true}^\frown\neg(\alpha^\frown\varphi^*)))$	by 4, $Mono_r$, 14
$14\ \neg(\mathrm{true}^\frown((\alpha^\frown\varphi^*{}^\frown\varphi)\wedge\neg(\alpha^\frown\varphi^*))$ $\vee(\varphi\wedge(\mathrm{true}^\frown\neg(\alpha^\frown\varphi^*))))$	by 13, N_r
$15\ \neg((\varphi^*\wedge\neg(\mathrm{true}^\frown\neg(\alpha^\frown\varphi^*))^\frown\varphi)\wedge$ $(\mathrm{true}^\frown\neg(\alpha^\frown\varphi^*)))$	by 11, 14
$16\ \varphi^*\Rightarrow\neg(\mathrm{true}^\frown\neg(\alpha^\frown\varphi^*)))$	by 9, 10, 15, $Mono_l$

17	$\varphi^* \wedge (\neg(\varphi^*{}^\frown\beta)^\frown\text{true}) \Rightarrow$ $((((\neg(\varphi^*{}^\frown\beta)^\frown\text{true}) \wedge \ell = 0)^\frown\text{true}) \vee$ $(((\varphi^* \wedge \neg(\neg(\varphi^*{}^\frown\beta)^\frown\text{true}))^\frown\varphi) \wedge$ $(\neg(\varphi^*{}^\frown\beta)^\frown\text{true}))^\frown\text{true})$	DC_3^*
18	$\ell = 0 \Rightarrow \beta$	assumption
19	$\ell = 0 \Rightarrow \varphi^*$	DC_1^*
20	$\ell = 0 \Rightarrow \neg(\neg(\varphi^*{}^\frown\beta)^\frown\text{true})$	18, 19, DC
21	$\neg((\neg(\varphi^*{}^\frown\beta)^\frown\text{true}) \wedge \ell = 0^\frown\text{true})$	20, DC
22	$\neg(((\varphi^* \wedge \neg(\neg(\varphi^*{}^\frown\beta)^\frown\text{true}))^\frown\varphi) \wedge (\neg(\varphi^*{}^\frown\beta)^\frown\text{true})^\frown\text{true})$	DC
23	$\varphi^* \Rightarrow \neg(\neg(\varphi^*{}^\frown\beta)^\frown\text{true})$	17, 20, 22, DC
24	$\beta \Rightarrow \neg(\text{true}^\frown\neg\gamma)$	assumption
25	$\varphi^* \Rightarrow \Box(\gamma \vee (\alpha^\frown\varphi^*{}^\frown\beta))$	16, 23, 24, DC

Now let us prove the correctness of the gas-burner design from the introduction as a last example of the working of our DC* axioms and rule. We have to give a derivation for

$$((\lceil\!\lceil Leak \rceil\!\rceil \wedge \ell \leq 1)^\frown(\lceil\!\lceil Nonleak \rceil\!\rceil \wedge \ell \geq 30))^* \Rightarrow \Box(\ell \geq 60 \Rightarrow 20 \textstyle\int Leak \leq \ell)\,.$$

Let

$\varphi \rightleftharpoons \lceil\!\lceil Leak \rceil\!\rceil \wedge \ell \leq 1^\frown\lceil\!\lceil \neg Leak \rceil\!\rceil \wedge \ell \geq 30$,

$\alpha \rightleftharpoons \ell = 0 \vee \lceil\!\lceil \neg Leak \rceil\!\rceil \vee (\lceil\!\lceil Leak \rceil\!\rceil \wedge \ell \leq 1^\frown\lceil\!\lceil \neg Leak \rceil\!\rceil \wedge \ell \geq 30)$,

$\beta, \gamma \rightleftharpoons \ell = 0 \vee (\ell \leq 1 \wedge \lceil\!\lceil Leak \rceil\!\rceil^\frown\ell = 0 \vee \lceil\!\lceil \neg Leak \rceil\!\rceil)$.

The formulas

$$\Box(\varphi \Rightarrow \neg(\text{true}^\frown\neg\alpha) \wedge \neg(\neg\beta^\frown\text{true})),\ \Box(\ell = 0 \Rightarrow \alpha \wedge \beta) \text{ and } \beta \Rightarrow \neg(\text{true}^\frown\neg\gamma)$$

are valid in DC without iteration. Therefore we can complete our derivation using (5), provided we can derive

$$\gamma \Rightarrow 20 \textstyle\int Leak \leq \ell \text{ and } (\alpha^\frown\varphi^*{}^\frown\beta) \wedge \ell \geq 60 \Rightarrow 20 \int Leak \leq \ell.$$

The first formula is straightforward to derive without DC*-specific axioms. Here follows a derivation for the second formula:

1	$\alpha \Rightarrow 31 \int Leak \leq \ell$	DC
2	$\varphi^* \wedge 31 \int Leak > \ell \Rightarrow (\varphi^* \wedge 31 \int Leak > \ell^\frown\text{true})$	DC
3	$\varphi \Rightarrow 31 \int Leak \leq \ell$	DC
4	$(\varphi^* \wedge 31 \int Leak > \ell^\frown\text{true}) \Rightarrow$ $(\ell = 0 \wedge 31 \int Leak > \ell^\frown\text{true}) \vee$ $(((\varphi^* \wedge 31 \int Leak \leq \ell^\frown\varphi) \wedge 31 \int Leak > \ell)^\frown\text{true})$	by DC^*3
5	$\ell = 0 \Rightarrow 31 \int Leak \leq \ell$	DC
6	$(\varphi^* \wedge 31 \int Leak > \ell^\frown\text{true}) \Rightarrow$ $(((\varphi^* \wedge 31 \int Leak \leq \ell^\frown\varphi) \wedge 31 \int Leak > \ell)^\frown\text{true})$	by 4, 5, $Mono_r$
7	$(\varphi^* \wedge 31 \int Leak \leq \ell^\frown\varphi) \Rightarrow 31 \int Leak \leq \ell$	by 2, 3, DC
8	$\varphi^* \Rightarrow 31 \int Leak \leq \ell$	by 6, 7, $Mono_r$
9	$(\alpha^\frown\varphi^*) \Rightarrow 31 \int Leak \leq \ell$	by 1, 8, DC
10	$\beta \Rightarrow \int Leak \leq 1$	DC
11	$(\alpha^\frown\varphi^*{}^\frown\beta) \wedge \ell \geq 60 \Rightarrow 20 \int Leak \leq \ell$	by 9, 10, DC

Iteration is known *chop-star* in Moszkowski's original discrete-time *ITL*, where it is regarded as part of the basic system. One should notice here that the axioms DC*1–DC*4 are valid in discrete-time *ITL* too and can be derived in its proof system. The rule DC*5, however, is new and DC-specific.

5 Hybrid Duration Calculus (*HDC*)

The modalities of DC are defined in terms of the chop-modality only. Hence, for every expressible modality only subintervals, of a given interval, can be reached. We also say that DC supports *contracting modalities*. In this section we study a hybrid version of *RDC** called *hybrid duration calculus* (*HDC*). *HDC* was introduced in [8], and the results of this section comes from that paper.

"Pure" modal logics do not have a mean for referencing worlds (in our case intervals) explicitly in formulas. In hybrid modal logics worlds can be referred to in the syntax. We shall see in following, that the hybrid machinery, in the case of *RDC*, gives us increased expressiveness as expanding modalities, for example, can be expressed. Furthermore, satisfiability for *HDC* is still decidable, and that problem is still non-elementary.

5.1 Syntax of *HDC*

We extend the language of RDC* (see Sect. 3.3) with a countable collection of symbols called *nominals*. We use $a, b, \ldots$ to range over nominals. A nominal will name one and only one interval. Furthermore, we extend the language with a satisfaction operator a: for each nominal a, with the global modality E and with the down-arrow binder $\downarrow$. The grammar of *HDC* is as follows:

$$\phi ::= P \mid \pi \mid \neg\phi \mid \phi \vee \psi \mid \phi^\frown \psi \mid a \mid a : \phi \mid E\phi \mid \downarrow a.\phi \,.$$

The intuition with the new types of formulas are: The formula a holds at the specific interval named by a only; the formula $a : \phi$ holds if ϕ holds on the interval named by a; $E\phi$ holds if there is some interval where ϕ holds; and $\downarrow a.\phi$ holds if ϕ holds under the assumption that a names the current interval. To limit the number of required parentheses, we will use the following precedence relation on the connectives: the down-arrow binders $\downarrow a$ have the lowest precedence; $^\frown$, $\vee$ and $\wedge$ have the next lowest precedence; $\neg$, E and the satisfaction operators $a :$ have the highest precedence.

5.2 Semantics of *HDC*

In order to give semantics for *HDC*, we introduce the notion of an *assignment* G that associates a unique interval $[t_a, u_a] \subseteq \mathbb{R}_{\geq 0}$ with each nominal a. An *interpretation* $\mathcal{I}$ for *HDC* is simply as for DC. For interpretations $\mathcal{I}$, assignments

G, intervals $[t,u] \subseteq \mathbb{R}_{\geq 0}$, and *HDC* formulas ϕ, we define the semantic relation $\mathcal{I}, G, [t,u] \models \phi$ below. We just give the cases which are special for *HDC*:

$$
\begin{array}{lll}
\mathcal{I}, G, [t,u] \models \pi & \text{iff} & u = t \\
\mathcal{I}, G, [t,u] \models P & \text{iff} & u > t \text{ and } P_{\mathcal{I}}(t) = 1 \text{ almost everywhere on } [t,u] \\
\mathcal{I}, G, [t,u] \models a & \text{iff} & G(a) = [t,u] \\
\mathcal{I}, G, [t,u] \models a : \phi & \text{iff} & \mathcal{I}, G, G(a) \models \phi \\
\mathcal{I}, G, [t,u] \models E\phi & \text{iff} & \text{for some interval } [v,w]\text{: } \mathcal{I}, G, [v,w] \models \phi \\
\mathcal{I}, G, [t,u] \models \downarrow a.\phi & \text{iff} & \mathcal{I}, G[\, a := [t,u]\,], [t,u] \models \phi\ ,
\end{array}
$$

where $G[\, a := [t,u]\,]$ is the assignment that assigns $[t,u]$ to a and agrees with G on all other nominals.

5.3 Expressivity of HDC

We will now give examples showing the extra expressiveness of *HDC*.

Propositional neighborhood logic: The interval logic *Neighbbourhood Logic* was introduced in [68]. Neighbourhood Logic has two basic modalities $\Diamond_l$ (reads: "for some left neighbourhood) and $\Diamond_r$ (reads: "for some right neighbourhood). These two modalities, which are both expanding, are defined by:

$$
\begin{array}{lll}
\mathcal{I}, [t,u] \models \Diamond_l \phi & \text{iff} & \mathcal{I}, [s,t] \models \phi \text{ for some } s \leq t \\
\mathcal{I}, [t,u] \models \Diamond_r \phi & \text{iff} & \mathcal{I}, [u,v] \models \phi \text{ for some } v \geq u,
\end{array}
$$

and they can be expressed in *HDC* in the following way:

$$
\begin{array}{l}
\Diamond_l \phi \text{ is defined by } \downarrow a.E(\phi \frown a) \\
\Diamond_r \phi \text{ is defined by } \downarrow a.E(a \frown \phi)\ ,
\end{array}
$$

where the correctness is easily checked using the semantics. This shows *HDC* to be more expressive than standard *RDC*, since *RDC* cannot express the neighborhood modalities or other expanding modalities.

Allen's interval relations: Allen has shown [2], that there are 13 possible relations between a pair of intervals. All these relations can, in *HDC*, be expressed in a natural manner [8]. Allen's interval relations are presented in Fig. 2, and, in Fig. 3, we show how each of the 13 relations can be expressed in *HDC*. In the formulas, a and b are nominals denoting the two intervals in question. It is a simple exercise to check the correctness of the translations in Fig. 3.

5.4 Monadic second-order theory of order

We shall reduce satisfiability of *HDC* to satisfiability of *monadic second-order theory of order*. The following presentation of monadic second-order theory of order, named $L_2^<$, is based on [53].

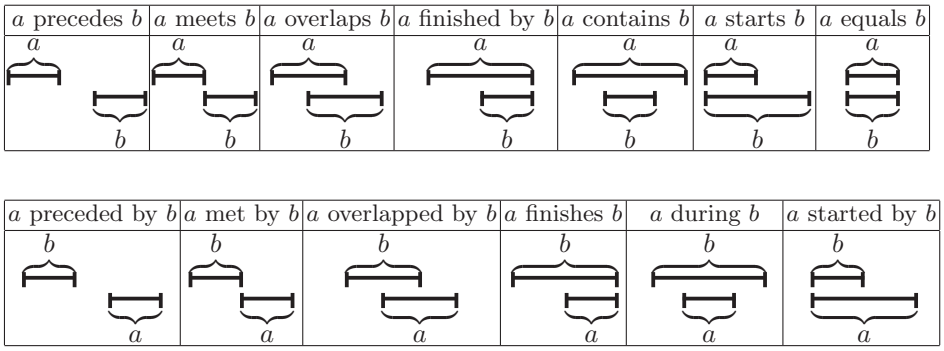


Fig. 2. The 13 possible relations between two intervals a and b.

a precedes b	$a : \Diamond_r(\neg\pi \wedge \Diamond_r b)$
a meets b	$a : \Diamond_r b$
a overlaps b	$E(\downarrow c.\neg\pi \wedge a : (\neg\pi \frown c) \wedge b : (c \frown \neg\pi))$
a finished by b	$a : (\neg\pi \frown b)$
a contains b	$a : (\neg\pi \frown b \frown \neg\pi)$
a starts b	$b : (a \frown \neg\pi)$
a equals b	$a : b$
a preceded by b	$a : \Diamond_l(\neg\pi \wedge \Diamond_l b)$
a met by b	$a : \Diamond_l b$
a overlapped by b	$E(\downarrow c.\neg\pi \wedge b : (\neg\pi \frown c) \wedge a : (c \frown \neg\pi))$
a finishes b	$b : (\neg\pi \frown a)$
a during b	$b : (\neg\pi \frown a \frown \neg\pi)$
a started by b	$a : (b \frown \neg\pi)$

Fig. 3. Representation in *HDC* of the 13 possible relations between intervals.

Syntax of $L_2^<$: The formulas are constructed from first-order variables, ranged over by $x, y, z, \ldots$, and second-order variables, ranged over by $P, Q, R, \ldots$, as described by the following grammar:

$$\phi ::= x < y \mid x \in P \mid \phi \vee \psi \mid \neg\phi \mid \exists x\phi \mid \exists P\phi \; .$$

Semantics of $L_2^<$: A *structure* $K = (A, B, <)$ *for* $L_2^<$ consists of a set A, partially ordered by $<$, and a set B of Boolean-valued functions on A. An element $b \in B$ can be considered a, possibly infinite, subset of A: $\{a \in A : b(a) = \text{true}\}$.

An *interpretation* $\mathcal{I}$ associates a member $P_\mathcal{I}$ of B to every second-order variable P and a *valuation* ν is a function assigning a member $\nu(x)$ of A to every

first-order variable x. The semantic relation $\mathcal{I}, \nu \models \phi$ is defined by:

$$\begin{array}{ll}
\mathcal{I}, \nu \models x < y & \text{iff } \nu(x) < \nu(y) \\
\mathcal{I}, \nu \models x \in P & \text{iff } \nu(x) \in P_{\mathcal{I}} \\
\mathcal{I}, \nu \models \neg\phi & \text{iff } \mathcal{I}, \nu \not\models \phi \\
\mathcal{I}, \nu \models \phi \vee \psi & \text{iff } \mathcal{I}, \nu \models \phi \text{ or } \mathcal{I}, \nu \models \psi \\
\mathcal{I}, \nu \models \exists x \phi & \text{iff for some } a \in A\text{: } \mathcal{I}, \nu[x := a] \models \phi \\
\mathcal{I}, \nu \models \exists P \phi & \text{iff for some } b \in B\text{: } \mathcal{I}[P := b], \nu \models \phi\ .
\end{array}$$

In the following we will assume that we have standard abbreviations for derived relations, propositional connectives and quantifiers. Furthermore, we use the following abbreviations:

$$x \geq y \mathrel{\widehat{=}} \neg(x < y) \qquad \text{and} \qquad x = y \mathrel{\widehat{=}} x \geq y \wedge y \geq x\ ,$$

as we will just consider sets A with a linear order.

We shall exploit the following two decidability results for $L_2^<$ to obtain our decidability results for discrete and continuous-time *HDC*.

The first result is a classical result by Buchi. Let ω denote the $L_2^<$ structure $(\mathbb{N}, 2^{\mathbb{N}}, <)$, where $2^{\mathbb{N}}$ denotes the set of Boolean-valued functions on natural numbers. The logic $L_2^<$ interpreted over the structure ω will be denoted $L_2^<(\omega)$.

Theorem 7. *[10] $L_2^<(\omega)$ is decidable.*

The second result is by Rabinovich [53] for so-called signal structures corresponding to interpretations of continuous-time DC. A Boolean-valued function h from $\mathbb{R}_{\geq 0}$ is called a *signal* [53] if there exists an unbounded increasing sequence $\tau_0 = 0 < \tau_1 < \tau_2 < \cdots < \tau_n < \cdots$ such that h is constant on every open interval $]\tau_i, \tau_{i+1}[, i \geq 0$. Let *SIGNAL* denote the set of all signals, and let *Sig* denote the signal structure $(\mathbb{R}_{\geq 0}, \mathit{SIGNAL}, <)$. The logic $L_2^<$ interpreted over the structure *Sig* will be denoted $L_2^<(\mathit{Sig})$.

Theorem 8. *[53] $L_2^<(\mathit{Sig})$ is decidable.*

5.5 Translation of *HDC* to $L_2^<$

The translation is strongly inspired by the translation of Quantified Discrete-Time DC to Monadic Logic over Finite Words, which is used in the tool DC-VALID [50]. The language of second order theory of one successor (called S1S) is $L_2^<$ extended by the successor function. For the structure ω, the successor function is definable in $L_2^<$, while for continuous structures S1S is more expressive, e.g. the validity of S1S is undecidable for signal structures [53]. In WS1S the interpretations of the second-order variables are restricted to finite sets. Since we have the global modality E where intervals of arbitrary size can be reached from any given interval we must base our results on $L_2^<$.

The translation – discrete time: In the translation, each state variable P corresponds to a second-order variable denoted by P also. The intuition with the formula $i \in P$ is that in the *HDC* interpretation $P(t) = 1$ in the interval $]i, i+1[$. Furthermore, for each nominal a we associate two unique first-order variables x_a and y_a, where the intuition is that a names the interval $[x_a, y_a]$.

The translation is defined recursively with respect to two first-order variables x and y naming the current interval $[x, y]$. These must be distinct from all the variables of the form x_a and y_a, where a is a nominal:

$$\begin{array}{ll}
\mathcal{T}_{x,y}(\pi) & = x = y \\
\mathcal{T}_{x,y}(P) & = x < y \wedge \forall z(x \leq z < y \rightarrow z \in P) \\
\mathcal{T}_{x,y}(\neg\phi) & = \neg\mathcal{T}_{x,y}(\phi) \\
\mathcal{T}_{x,y}(\phi \vee \psi) & = \mathcal{T}_{x,y}(\phi) \vee \mathcal{T}_{x,y}(\psi) \\
\mathcal{T}_{x,y}(\phi \frown \psi) & = \exists z(\mathcal{T}_{x,z}(\phi) \wedge \mathcal{T}_{z,y}(\psi) \wedge x \leq z \wedge z \leq y) \\
\mathcal{T}_{x,y}(a) & = x = x_a \wedge y = y_a \\
\mathcal{T}_{x,y}(a : \phi) & = \mathcal{T}_{x_a,y_a}(\phi) \\
\mathcal{T}_{x,y}(E\phi) & = \exists x \exists y(x \leq y \wedge \mathcal{T}_{x,y}(\phi)) \\
\mathcal{T}_{x,y}(\downarrow a.\phi) & = \exists x_a \exists y_a(x = x_a \wedge y = y_a \wedge \mathcal{T}_{x,y}(\phi)) \ .
\end{array}$$

In the translation for chop we assume that z is a "fresh" variable and distinct from x_a and y_a for all nominals a.

The translation – continuous time: In the continuous-time case, the formulas of $L_2^<$ are interpreted in signal structures and [53] contains a translation from *RDC* to $L_2^<$. We can adapt the translation above to a translation for continuous-time *HDC* by changing the translation of P only. The translation $\mathcal{T}_{x,y}(P)$ is:

$$\begin{array}{l}
x < y \wedge \forall z(x < z < y \rightarrow \exists v(x < v < z \wedge \forall t(v < t < z \rightarrow t \in P))) \\
\qquad \wedge \forall z(x < z < y \rightarrow \exists v(z < v < y \wedge \forall t(z < t < v \rightarrow t \in P))) \ .
\end{array}$$

The idea behind this translation is: P is 1 almost everywhere in $[x, y]$ $(x < y)$ iff for every $z \in]x, y[$ there are left and right neighborhoods of z where P is constant and equal to 1.

We will just refer to [8] for the correctness proofs for the translations. The main result only is stated here.

Theorem 9. *[8] HDC is decidable for discrete and continuous-time domains.*

Since neighbourhood modalities are expressible in *HDC* this result shows that a propositional neighbourhood logic based on *RDC* is decidable.

6 Model-checking: Using Priced Timed Automata

In this section we will present the model-checking result described in [16]. In particular, we shall address the model-checking problem:

$$A \models \neg\phi \ ,$$

where A is an arbitrary timed automaton. The formula ϕ is a negation-free formulas where linear duration constraints can occur in an arbitrary positive Boolean context as described by the grammar:

$$S ::= 0 \mid 1 \mid P \mid \neg S \mid S_1 \vee S_2$$
$$\phi ::= \ell \bowtie k \mid [\![S]\!] \mid \sum_{i=1}^{m} c_i \int S_i \bowtie k \mid \phi \vee \psi \mid \phi \wedge \psi \mid \phi \frown \psi \ ,$$

where $k, m, c_i \in \mathbb{N}$, and $\bowtie \in \{<, \leq, =, \geq, >\}$. We shall assume that formulas contain only upper-bound constraints on durations, i.e. where $\bowtie \in \{<, \leq\}$, and where exactly one duration constraint is a strict inequality. Such formulas are called *negation-free formula* in this section. Adding general negation to this fragment would lead to one of the undecidable fragments described in Sect. 3.

In a model-checking problem of the form $A \models \neg\phi$, the formula ϕ is a specification of an undesired situation – a counter example – and $A \models \neg\phi$ expresses that no run of A exist where this undesired situation occurs. This idea is, for example, pursued in [47, 30], where ϕ can have the restricted form of a *DC implementable* [57], thus abandoning accumulated durations and replacing chop by more restricted, operationally inspired operators.

6.1 Multi-Priced Timed Automata

We shall reduce the satisfiability problem for negation-free formulas above to a problem of computing minimal costs in *multi-priced timed automata* (MPTA) [41]. MPTA are an extension of timed automata [3, 4], where *prices* are associated with edges and locations. The cost of taking an edge is the price of that edge, and the cost of staying in a location is given by the product of the *cost-rate* for that location and the time spent in the location.

Let $\mathbb{C}$ be a finite set of clocks. An *atomic constraint* is a formula of the form: $x \bowtie n$, where $x \in \mathbb{C}$, $\bowtie \in \{\leq, =, \geq, <, >\}$, and $n \in \mathbb{N}$. A *clock constraint* over $\mathbb{C}$ is a conjunction of atomic constraints. Let $B(\mathbb{C})$ denote the set of clock constraints over $\mathbb{C}$ and let $B(\mathbb{C})^*$ denote the set of clock constraints over $\mathbb{C}$ involving only upper bounds, i.e. $\leq$ or $<$. Furthermore, let $2^{\mathbb{C}}$ denote the power set of $\mathbb{C}$.

A *clock valuation* $v : \mathbb{C} \to R_{\geq 0}$ is a function assigning a non-negative real number with each clock. The valuation v *satisfies* a clock constraint $g \in B(\mathbb{C})$, if each conjunct of g is true in v. In this case we write $v \in g$. Let $\mathbb{R}^{\mathbb{C}}_{\geq 0}$ denote the set of all clock valuations.

Definition 1. *([41]) A* multi-priced timed automaton A *over clocks* $\mathbb{C}$ *is a tuple* (L, l_0, E, I, P)*, where* L *is a finite set of locations,* l_0 *is the initial location,* $E \subseteq L \times B(\mathbb{C}) \times 2^{\mathbb{C}} \times L$ *is the set of edges, where an edge contains a source, a guard, a set of clocks to be reset, and a target.* $I : L \to B(\mathbb{C})^*$ *assigns invariants to locations, and* $P : (L \cup E) \to \mathbb{N}^m$ *assigns a vector of prices to both locations and edges. In the case of* $(l, g, r, l') \in E$*, we write* $l \xrightarrow{g,r} l'$*.*

A *multi-priced transition system* is a structure $T = (S, s_0, \Sigma, \longrightarrow)$, where S is a, possibly infinite, set of states, $s_0 \in S$ is the initial state, Σ is a finite set of labels, and $\longrightarrow$ is a partial function from $S \times \Sigma \times S$ to $\mathbb{R}^m_{\geq 0}$, defining the

possible transitions and their associated costs. The notation $s \xrightarrow{a}_{\boldsymbol{p}} s'$ means that $\longrightarrow (s, a, s')$ is defined and equal to $\boldsymbol{p}$.

An *execution* of T is a finite sequence $\alpha = s_0 \xrightarrow{a_1}_{\boldsymbol{p}_1} s_1 \cdots s_{n-1} \xrightarrow{a_n}_{\boldsymbol{p}_n} s_n$, and the associated *cost* of α is $\text{cost}(\alpha) = \sum_{i=1}^{n} \boldsymbol{p}_i$.

For a given state s and a vector $\boldsymbol{u} = (u_1, \ldots, u_{m-1}) \in \mathbb{R}^{m-1}_{\geq 0}$, let $\text{mincost}_{T,\boldsymbol{u}}(s)$ denote the *minimum cost* wrt. the last component of the price vector of reaching s while respecting the upper bound constraints to the other prices which are given by $\boldsymbol{u}$. This is defined as the infimum of the cost of all executions ending in s and respecting price constraint $\boldsymbol{u}$, i.e.

$$\text{mincost}_{T,\boldsymbol{u}}(s) = \inf \left\{ \text{cost}(\alpha)_m \;\middle|\; \begin{array}{l} \alpha \text{ an execution of } T \text{ ending in } s, \\ \forall i \in \mathbb{N}_{<m}.\text{cost}(\alpha)_i \leq u_i \end{array} \right\} .$$

Furthermore, for a set of states $G \subseteq S$, let $\text{mincost}_{T,\boldsymbol{u}}(G)$ denote the minimal cost of reaching some state in G while respecting the upper price bounds $\boldsymbol{u}$.

The semantics of a linearly multi-priced timed automaton $A = (L, l_0, E, I, P)$ is a *multi-priced transition system* $T_A = (S, s_0, \Sigma, \longrightarrow)$, where

- $S = L \times \mathbb{R}^{\mathbb{C}}_{\geq 0}$,
- $s_0 = (l_0, v_0)$, where v_0 is the (clock) valuation assigning 0 to every clock,
- $\Sigma = E \cup \{\delta\}$, where δ indicates a delay and $e \in E$ the edge taken, and
- the partial transition function $\longrightarrow$ is defined as follows:
  - $(l, v) \xrightarrow{\delta}_{\boldsymbol{p}} (l, v + d)$ if $\forall e.0 \leq e \leq d : v + e \in I(l)$, and $\boldsymbol{p} = d \cdot P(l)$,
  - $(l, v) \xrightarrow{e}_{\boldsymbol{p}} (l', v')$ if $(l, g, r, l') \in E, v \in g, v' = v[r \mapsto 0]$ and $\boldsymbol{p} = P(e)$,

  where $v + d$ means the clock valuation where the value of x is $v(x) + d$, for $x \in \mathbb{C}, d \in R_{\geq 0}$, and $v[r \mapsto 0]$ is the valuation which is as v except that clocks in r are mapped to 0.

In case T_A performs a δ step $(l, v) \xrightarrow{\delta}_{\boldsymbol{p}} (l, v + d)$, we say that the *duration of the step* is d. All other steps, i.e. those labelled $e \in E$, have duration 0.

The main results that we shall exploit concerning linearly multi-priced timed automata is that the minimum cost of reaching some target location is computable for any (set of) target location(s) and any upper bound on the remaining prices: Given an MPTA $A = (L, l_0, E, I, P)$, a target $G \subset L$, and some cost constraint $\boldsymbol{u} \in \mathbb{R}^{m-1}_{\geq 0}$, we define the *minimum cost* $\text{mincost}_{A,\boldsymbol{u}}(G)$ to be $\text{mincost}_{T_A,\boldsymbol{u}}(G \times \mathbb{R}^{\mathbb{C}}_{\geq 0})$.

Theorem 10. *[41] For any MPTA $A = (L, l_0, E, I, P)$, any set $G \subset L$, and any cost constraint $\boldsymbol{u} \in \mathbb{R}^{m-1}_{\geq 0}$, the minimum cost* $\text{mincost}_{A,\boldsymbol{u}}(G)$ *is computable.*

6.2 Representing Negation-Free Formulas by MPTA

In the construction of a MPTA for a negation-free formula ϕ, we represent ϕ by a tuple $(L, s, E, I, P, f, \Lambda)$ denoted A_ϕ, where (L, s, E, I, P) is a multi-priced timed automaton, f is a special *final location* to be reached, and Λ is a function associating a state expression S (of DC) with every location. The automaton will

not spend (positive) time in the final location, and the idea is that the runs of A_ϕ from the start location s to f represent the models of ϕ. Durational constraints $\sum_{i=1}^{m} c_i \int S_i \bowtie k$ are, however, treated in a special way: for any run from s to f, the value of $\sum_{i=1}^{m} c_i \int S_i$ is the cost of the execution, and an analysis of minimal costs will be used decide the satisfaction of $\sum_{i=1}^{m} c_i \int S_i \bowtie k$.

The construction [16]: We shall use the following conventions: the cost of an edge is always 0, the cost-rate of a location is 0 unless otherwise stated, the invariant of a location is true unless otherwise stated, the mark of a location l is the state expression 1, i.e. $\Lambda(l) = 1$, unless otherwise stated. Furthermore, we assume that the formula ϕ contains n distinct state variables $P_1, \ldots, P_n$ and m subformulas $\sum_{i=1}^{m_j} c_{i,j} \int S_{i,j} \bowtie_j k_j$, where $\bowtie_m = <$ and $\bowtie_j = \leq$ for every $j < m$. The construction follows the structure of the formula.

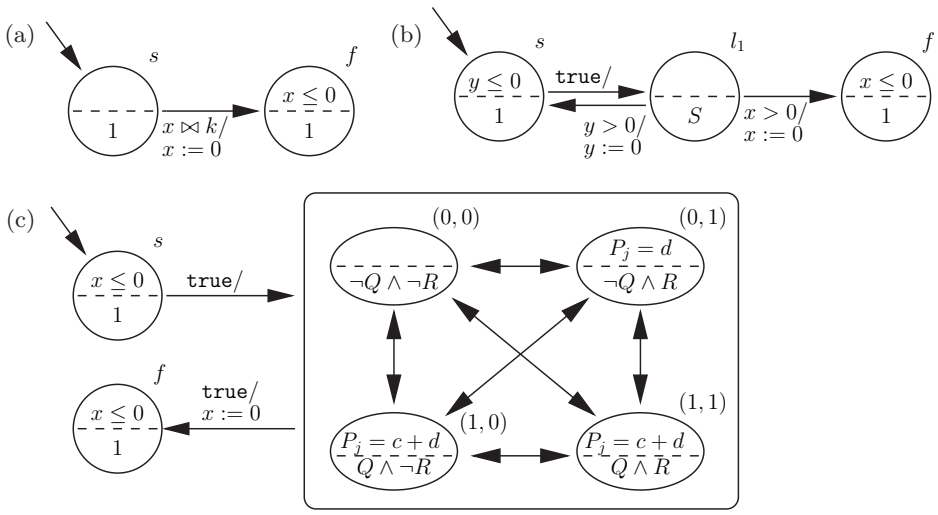


Fig. 4. MPTA encoding of atomic formulas: (a) $\ell \bowtie k$, (b) $\lceil S \rceil$, (c) $c \int Q + d \int Q \vee R \bowtie k$. State decorations above the dashed line denote invariants and cost assignments (both omitted if trivial), while those below the dashed line denote the labeling function Λ.

The case $\phi = \ell \bowtie k$. (See Fig. 4(a).) Let $A_\phi = (L, s, E, I, P, f, \Lambda)$, where

- $L = \{s, f\}$,
- $E = \{(s, x \bowtie k, \{x\}, f)\}$, and
- $I(f) = x \leq 0$.

The case $\phi = \lceil S \rceil$. (See Fig. 4(b).) Let $A_\phi = (L, s, E, I, P, f, \Lambda)$, where

- $L = \{s, l_1, f\}$,

– $E = \{e_1, e_2, e_3\}$, where $e_1 = (s, \text{true}, \{\}, l_1)$, $e_2 = (l_1, y > 0, \{y\}, s)$, and $e_3 = (l_1, x > 0, \{x\}, f)$,
– $I(s) = y \leq 0$ and $I(f) = x \leq 0$, and
– $\Lambda(l_1) = S$.

The case $\phi = \sum_{i=1}^{m_j} c_{i,j} \int S_{i,j} \bowtie_j k_j$. (See Fig. 4(c).) Let $A_\phi = (L, s, E, I, P, f, \Lambda)$, where $L = \{s, f\} \cup \{0,1\}^n$ and E, I, P and Λ are defined below. Each bit-vector $\boldsymbol{b} = (b_1, \ldots, b_n) \in \{0,1\}^n$ is such that $b_i = 1$ iff P_i is 1 in that state. The edges $E = E_1 \cup E_2 \cup E_3$, where

– $E_1 = \{(s, \text{true}, \emptyset, \boldsymbol{b}) \mid \boldsymbol{b} \in \{0,1\}^n\}$,
– $E_2 = \{(\boldsymbol{b}, \text{true}, \emptyset, \boldsymbol{b'}) \mid \boldsymbol{b}, \boldsymbol{b'} \in \{0,1\}^n \wedge \boldsymbol{b} \neq \boldsymbol{b'}\}$, and
– $E_3 = \{(\boldsymbol{b}, \text{true}, \{x\}, f) \mid \boldsymbol{b} \in \{0,1\}^n\}$.

For $\boldsymbol{b} \in \{0,1\}^n$, we define two sets: $\boldsymbol{b}^+ = \{l \in \mathbb{N} \mid 1 \leq l \leq n \wedge b_l = 1\}$ and $\boldsymbol{b}^- = \{l \in \mathbb{N} \mid 1 \leq l \leq n \wedge b_l = 0\}$. Let $F(\boldsymbol{b})$ denote the state expression:

$$\bigwedge_{l \in \boldsymbol{b}^-} \neg P_l \wedge \bigwedge_{l \in \boldsymbol{b}^+} P_l \ .$$

For each state expression $S_{i,j}$ occurring in the summation $\sum_{i=1}^{m_j} c_{i,j} \int S_{i,j}$, we define the cost rate as follows:

$$C(\boldsymbol{b})(S_{i,j}) = \begin{cases} c_{i,j}, & \text{if } F(\boldsymbol{b}) \Rightarrow S_{i,j}, \\ C(\boldsymbol{b})(S_{i,j}) = 0 & \text{otherwise.} \end{cases}$$

The invariants of locations are as follows: $I(s) = x \leq 0, I(f) = x \leq 0$, and for all other locations the invariant is true.
The cost assignment $P : L \cup E \to \mathbb{N}^m$ is defined as follows:

$$P(l)_k = \begin{cases} 0 & \text{if } l = s \text{ or } l = f \text{ or } k \neq j \text{ or } l \in E \\ \sum_{i=1}^{m_j} C(l)(S_{i,j}) & \text{otherwise.} \end{cases}$$

The definition Λ is $\Lambda(l) = 1$ if $l = s$ or $l = f$, and $F(l)$ otherwise.

In the recursive cases: $\phi \vee \psi$, $\phi \wedge \psi$ and $\phi \frown \psi$ below, we will assume that $A_\phi = (L_1, s_1, E_1, I_1, P_1, f_1, \Lambda_1)$ and $A_\psi = (L_2, s_2, E_2, I_2, P_2, f_2, \Lambda_2)$, have disjoint sets of locations and clocks, respectively. These constructions are generalization of standard constructions on finite automata.

The case $\phi \vee \psi$. Assume that s and f are two new locations and that x is a new clock. Let $A_{\phi \vee \psi} = (L, s, E, I, P, f, \Lambda)$, where

– $L = \{s, f\} \cup L_1 \cup L_2$,
– $E = \{e_1, e_2, e_3, e_4\} \cup E_1 \cup E_2$, where $e_1 = (s, \text{true}, \{\}, s_1)$, $e_2 = (s, \text{true}, \{\}, s_2)$, $e_3 = (f_1, \text{true}, \{x\}, f)$, and $e_4 = (f_2, \text{true}, \{x\}, f)$.
– $I(s) = I(f) = x \leq 0$, $I(l) = I_1(l)$, for $l \in L_1$, and $I(l) = I_2(l)$, for $l \in L_2$,
– $P(l) = P_1(l)$, for $l \in L_1$, and $P(l) = P_2(l)$, for $l \in L_2$, and
– $\Lambda(l) = \Lambda_1(l)$, for $l \in L_1$, and $\Lambda(l) = \Lambda_2(l)$, for $l \in L_2$.

The case: $\phi \wedge \psi$. Let $A_{\phi\wedge\psi} = (L, (s_1, s_2), E, I, P, (f_1, f_2), \Lambda)$, where

- $L = \{(l_1, l_2) \in L_1 \times L_2 \mid \Lambda_1(l_1) \wedge \Lambda_2(l_2) \text{ is satisfiable}\}$,
- $E = \left\{ ((l_1, l_2), g_1 \wedge g_2, r_1 \cup r_2, (l_1', l_2')) \;\middle|\; \begin{array}{c} (l_1, l_2), (l_1', l_2') \in L \\ \wedge\, (l_1, g_1, r_1, l_1') \in E_1 \\ \wedge\, (l_2, g_2, r_2, l_2') \in E_2 \end{array} \right\} \cup$
  $\{((l_1, l_2), g_1, r_1, (l_1', l_2)) \mid (l_1, l_2), (l_1', l_2) \in L \wedge (l_1, g_1, r_1, l_1') \in E_1\} \cup$
  $\{((l_1, l_2), g_1, r_1, (l_1, l_2')) \mid (l_1, l_2), (l_1, l_2') \in L \wedge (l_2, g_2, r_2, l_2') \in E_2\}$
- $I(l_1, l_2) = I_1(l_1) \wedge I_2(l_2)$, for $(l_1, l_2) \in L$,
- $P(l_1, l_2)_k = P_1(l_1)_k + P_2(l_2)_k$, for $(l_1, l_2) \in L$ and $1 \le k \le m$ and
- $\Lambda(l_1, l_2) = \Lambda_1(l_1) \wedge \Lambda_2(l_2)$, for $(l_1, l_2) \in L$.

The case: $\phi \frown \psi$. Let $A_{\phi\frown\psi} = (L_1 \cup L_2, s_1, E, I, P, f_2, \Lambda)$, where

- $E = \{(f_1, \text{true}, C_2, s_2)\} \cup E_1 \cup E_2$, where C_2 is the set of clocks used by A_ψ,
- $I(l) = I_1(l)$, for $l \in L_1$, and $I(l) = I_2(l)$, for $l \in L_2$,
- $P(l) = P_1(l)$, for $l \in L_1$, and $P(l) = P_2(l)$, for $l \in L_2$.
- $\Lambda(l) = \Lambda_1(l)$, for $l \in L_1$, and $\Lambda(l) = \Lambda_2(l)$, for $l \in L_2$.

The transition from f_1 to s_2 is taken immediately when f_1 is reached, as the clock constraints in $I_1(f_1)$ does not permit durational stays in f_1.

In order to reduce the satisfiability problem for negation-free formulas to computation of minimal costs for MPTA, a correspondence between interpretations for formulas and runs of MPTA must be established: consider a transition $(l, v) \xrightarrow{\delta}_{\boldsymbol{p}} (l, v + d)$, where the location l is labelled with the state expression S. This transition corresponds to a set of interpretations (of length δ) for which S is 1 almost everywhere. The set of interpretations corresponding to an execution is obtained by concatenation of the interpretations for the individual transitions. For further details we just refer to [16], and assume for now that for every run α of A_ϕ, there is a set of *observations* $Intp(\alpha)$ corresponding to α, where an observation is a pair consisting of an interpretation and an interval. The correspondence between runs and observations is stated in the following lemma.

Lemma 2. *[16] Let ϕ be a negation-free formula. Then $\mathcal{I}, [0, e] \models \phi$ iff there exists a run α of A_ϕ with $(\mathcal{I}, [0, e]) \in Intp(\alpha)$ and $\text{cost}(\alpha)_j \bowtie_j k_j$ for $1 \le j \le m$.*

The main result follows from this lemma.

Theorem 11. *[16] Let ϕ be a negation-free formula, $A_\phi = (L, s, E, I, P, f, \Lambda)$, and $\boldsymbol{u} = (k_1, \ldots, k_{m-1})$. Then $\text{mincost}_{(L,s,E,I,P),\boldsymbol{u}}(f) < k_m$ iff ϕ is satisfiable.*

The above construction can also be used for model-checking a timed automaton wrt. a negation of a negation-free formula, as stated in the next theorem.

Theorem 12. *[16] Let $A = (L_1, s_1, E_1, I_1, \Lambda_1)$ be a timed automaton (L_1, s_1, E_1, I_1) extended by a location labelling $\Lambda_1 : L_1 \to S$, ϕ a negation-free formula, $A_\phi = (L_2, s_2, E_2, I_2, P_2, f_2, \Lambda_2)$, and $\boldsymbol{u} = (k_1, \ldots, k_{m-1})$.*
Then $A \models \neg\phi$ iff $\text{mincost}_{(L,s,E,I,P),\boldsymbol{u}}(f \times L_1) \ge k_m$, where

- *$B = (L_1, s_1, E_1, I_1, P_0, s, \Lambda_1)$ is A converted to an MPTA by extension with the trivial cost function $P_0 \equiv \mathbf{0}$ and an irrelevant terminal state $s \in L_1$,*
- *$(L, s, E, I, P, f, \Lambda) = A_\phi \otimes B$ is the MPTA-product from case $\phi \wedge \psi$.*

7 Model-checking: Linear Duration Invariants

In this section, we present a model checking technique for a class of chop-free formulas of DC called *linear duration invariants* which is different from the technique presented in the previous section. From now on in this chapter, we restrict ourselves to the class of DC formulas that do not have global (rigid) variables. Therefore, the valuation $\mathcal{V}$ is irrelevant in DC models and will be dropped out from them.

Let $\mathbf{S}$ ranged over by $s, u, v, \ldots$, be a finite set of state variables of the DC language. A linear duration invariant is a DC formula of the form

$$\psi \mathrel{\widehat{=}} (A \leq \ell \leq B \Rightarrow \textstyle\sum_{s \in \mathbf{S}} c_s \int s \leq M)$$

where A, B, c_s, M, $(A \leq B)$ are fixed real numbers (B may be ∞).

We address in this section the problem of checking if a timed automaton satisfies a linear duration invariant. To simplify our presentation and to make the problem easier, we restrict ourselves to the class of timed automata whose behaviour can be represented by a class of so-called Timed Regular Expressions (TRE) that has been introduced in [1] and also from our earlier work [34, 43]. The relationship between TRE's and timed automata was presented in [1] which says that the class of timed languages recognised by timed automata can be received from that of timed regular languages with renaming.

A DC model represents an observation of the behaviour of states in $\mathbf{S}$ in an interval of time. It consists of an interval $[0, T]$ and an interpretation $\mathcal{I}$ in the interval $[0, T]$ of the states in $\mathbf{S}$.

Timed Regular Expressions are defined as follows. For a TRE R, let $state(R)$ denote the set of states occurring in R.

Definition 2. TRE *are defined recursively by*

1. ϵ *is a* TRE *and* $state(\epsilon) = \emptyset$.
2. *For any* $s \in \boldsymbol{S}$, s *is a* TRE *and* $state(s) = \{s\}$.
3. *If* R *is a* TRE, *for any real numbers* a, b, $0 \leq a \leq b$ *(*b *may be* ∞*), the pair* $(R, [a, b])$ *is a* TRE, *and* $state(R, [a, b]) = state(R)$.
4. *If* R_1, R_2 *are* TRE*'s, then* R_1^*, $R_1 \frown R_2$, $R_1 \oplus R_2$ *are* TRE*'s, and* $state(R_1^*) = state(R_1)$; $state(R_1 \frown R_2) = state(R_1 \oplus R_2) = state(R_1) \cup state(R_2)$.
5. *If* R_1, R_2 *are* TRE*'s, and* $state(R_1) \cap state(R_2) = \emptyset$, *then* $R_1 \otimes R_2$ *is a* TRE, *and* $state(R_1 \otimes R_2) = state(R_1) \cup state(R_2)$.

Here we overload the operator $\frown$ to be defined in TRE's because the meaning we are going to give to it is similar to its meaning in DC. The operator $\frown$ is for sequential composition, the operator $\otimes$ for parallel composition, and $*$ for repetition.

To see how TRE's are used as abstract model for real-time systems, we associate a set of Duration Calculus model to each TRE. Each TRER defines a class of DC models $\mathcal{M}(R)$ as:

1. A model $\sigma = (\mathcal{I}, [0, T])$ is in $\mathcal{M}(\epsilon)$ iff $T = 0$.

2. A model $\sigma = (\mathcal{I}, [0,T])$ is in $\mathcal{M}(s)$ iff $0 \leq T$, and for all $t \in [0,T]$ $s_{\mathcal{I}}(t) = 1$, and for all $s' \neq s$ $s'_{\mathcal{I}}(t) = 0$.
3. A model $\sigma = (\mathcal{I}, [0,T])$ is in $\mathcal{M}(R, [a,b])$ iff $\sigma = (\mathcal{I}, [0,T]) \in \mathcal{M}(R)$ and $a \leq T \leq b$.
4. A model $\sigma = (\mathcal{I}, [0,T])$ is in $\mathcal{M}(R_1{}^\frown R_2)$ iff there are $0 \leq T' \leq T$, $\sigma_1 = (\mathcal{I}_1, [0,T']) \in \mathcal{M}(R_1)$, $\sigma_2 = (\mathcal{I}_2, [0, T-T']) \in \mathcal{M}(R_2)$ such that for all $s \in state(R_1) \cup state(R_2)$, $s_{\mathcal{I}_1}(t) = s_{\mathcal{I}}(t)$ for all $t \in [0,T')$ and $s_{\mathcal{I}_2}(t-T') = s_{\mathcal{I}}(t)$ for all $t \in [T',T]$, and for all $s' \notin state(R_1) \cup state(R_2)$, $s'_{\mathcal{I}}(t) = 0$ for all $t \in [0,T]$. We also define $\sigma = \sigma_1{}^\frown\sigma_2$.
5. A model $\sigma = (\mathcal{I}, [0,T])$ is in $\mathcal{M}(R_1 \otimes R_2)$ iff there are $\sigma_1 = (\mathcal{I}_1, [0,T]) \in \mathcal{M}(R_1)$, $\sigma_2 = (\mathcal{I}_2, [0,T]) \in \mathcal{M}(R_2)$ such that for all $t \in [0,T]$, $s_{\mathcal{I}_1}(t) = s_{\mathcal{I}}(t)$ for all $s \in state(R_1)$, and $s_{\mathcal{I}_2}(t) = s_{\mathcal{I}}(t)$ for all $s \in state(R_2)$, and $s'_{\mathcal{I}}(t) = 0$ for all $s' \notin state(R_1) \cup state(R_2)$, and then we define $\sigma_1 \otimes \sigma_2$ as σ.
6. A model $\sigma = (\mathcal{I}, [0,T]) \in \mathcal{M}(R_1 \oplus R_2)$ iff $\sigma \in \mathcal{M}(R_1)$ or $\sigma \in \mathcal{M}(R_2)$
7. A model $\sigma = (\mathcal{I}, [0,T]) \in \mathcal{M}(R^*)$ iff there is an integer $k \geq 0$ such that $\sigma \in \mathcal{M}(R^k)$, where $R^0 \mathrel{\widehat{=}} \epsilon$, and for $k > 0$, $R^k \mathrel{\widehat{=}} R^\frown R^{k-1}$. Or, equivalently, either $\sigma = (\mathcal{I}, [0,0])$ or there are models $\sigma_1, \ldots, \sigma_k \in \mathcal{M}(R)$ such that $\sigma = \sigma_1{}^\frown\sigma_2{}^\frown \ldots {}^\frown\sigma_k$. (It should be noted here that $^\frown$ is associative).

So, if we consider R as the abstract model of a real-time system, a model $\sigma \in \mathcal{M}(R)$ represents a behaviour of the system. Our checking problem in this subsection is to decide if $\sigma \models \psi$ for all $\sigma \in \mathcal{M}(R)$ for a given TRE R and a given LDI ψ.

The algorithm for solving this problem is presented via a series of lemmas and theorems. Therefore, for the readability, we also give a proof for some theorems in this section.

For simplicity, let $d_s(\sigma)$ denote the accumulated time (duration) of state $s \in \mathbf{S}$ over the interval $[0,T]$ under the interpretation $\mathcal{I}$, i.e. $d_s(\sigma) = \int_0^T s_{\mathcal{I}}(t)dt$, let $d(\sigma)$ denote the length of the interval $[0,T]$, i.e. T, and let $inv(\sigma)$ denote $\sum_{s \in \mathbf{S}} c_s d_s(\sigma)$ (i.e. $\sum_{s \in \mathbf{S}} c_s \int s$ evaluated over σ). Hence, $\sigma \models \psi$ iff $A \leq d(\sigma) \leq B \Rightarrow inv(\sigma) \leq M$.

Lemma 3. *[34] Let σ, $\sigma_1 = (\mathcal{I}_1, [0,T_1])$, $\sigma_2 = (\mathcal{I}_2, [0,T_2]) \in \mathcal{M}(R_2)$ be DC models. Then,*

1. *if $\sigma = \sigma_1{}^\frown\sigma_2$ then $d_s(\sigma) = d_s(\sigma_1) + d_s(\sigma_2)$ for all $s \in \boldsymbol{S}$, $d(\sigma) = d(\sigma_1) + d(\sigma_2)$ and $inv(\sigma) = inv(\sigma_1) + inv(\sigma_2)$, and*
2. *if $\sigma = \sigma_1 \otimes \sigma_2$ then $d_s(\sigma) = d_s(\sigma_1) + d_s(\sigma_2)$ for all $s \in \boldsymbol{S}$, $d(\sigma) = d(\sigma_1) = d(\sigma_2)$, and $inv(\sigma) = inv(\sigma_1) + inv(\sigma_2)$.*

We will not distinguish the TRE's that define the same set of DC models.

Definition 3. *For arbitrary TRE's R_1, R_2, we say $R_1 \equiv R_2$ iff $\mathcal{M}(R_1) = \mathcal{M}(R_2)$.*

The following theorem can be proved by direct check.

Theorem 13. *For arbitrary TRE's R, R_1, R_2*

1. *$(R_1 \oplus R_2)^\frown R \equiv (R_1{}^\frown R) \oplus (R_2{}^\frown R)$ and $R^\frown(R_1 \oplus R_2) \equiv (R^\frown R_1) \oplus (R^\frown R_2)$*

2. $(R_1 \oplus R_2) \otimes R \equiv (R_1 \otimes R) \oplus (R_2 \otimes R)$ *and* $R \otimes (R_1 \oplus R_2) \equiv (R \otimes R_1) \oplus (R \otimes R_2)$
3. $(R_1 \oplus R_2)^* \equiv ((R_1^*)^\frown(R_2^*))^*$

Theorem 13 implies that any TRE $\mathcal{R}$ can be written as $\mathcal{R}_1 \oplus \mathcal{R}_2 \oplus \ldots \oplus \mathcal{R}_k$, where each $\mathcal{R}_i$ is a TRE in which there is no occurrence of $\oplus$.

Since we are interested in checking a TRE for the linear duration invariant ψ, we will not distinguish the TRE's of which the sets of models satisfy ψ at the same time, and define:

Definition 4. *R_1 and R_2 are ψ-equivalent, denoted by $R_1 \equiv_\psi R_2$, iff $R_1 \models \psi$ if and only if $R_2 \models \psi$.*

Of course, if $R_1 \equiv R_2$ then $R_1 \equiv_\psi R_2$. The following theorem follows immediately from Lemma 3.

Theorem 14. *For arbitrary TRE's R_1, R_2*

1. $R_1{}^\frown R_2 \equiv_\psi R_2{}^\frown R_1$
2. $(R_1 \oplus R_2)^* \equiv_\psi (R_1^*)^\frown(R_2^*)$
3. $((R_1^*)^\frown R_2)^* \equiv_\psi (R_1^*)^\frown R_2^*$
4. $(R_1^* \otimes R_2^*)^* \equiv (R_1^* \otimes R_2^*)$

Definition 5.

- *A TRE R in which there is no occurrence of the operator $*$ is said to be finite. Otherwise, R is said to be infinite.*
- *A simple TRE is a finite TRE in which there is no occurrence of the operator $\oplus$.*

First we show how to decide for a finite TRE R, $R \models \psi$. We will only have to consider the simple TRE's because any finite TRE R can be written as $R_1 \oplus \ldots \oplus R_k$, where each R_i is a simple TRE's, and $R \models \psi$ iff for all i $(i \leq k)$, $R_i \models \psi$.

Let R be a simple TRE. We associate a set of linear constraints $\mathcal{C}(R)$, the set of durations $\{d_s(R) \mid s \in \mathbf{S}\}$ and execution time $d(R)$ to R as follows. Let $Var(\mathcal{C}(R))$ denote the set of (real) variables occurring in $\mathcal{C}(R)$.

Definition 6.

- *Let $R = s$. Then $\mathcal{C}(R) = \emptyset$, $d_s(R) = t$, $d(R) = t$ where t is a real variable, and $d_{s'}(R) = 0$ for all $s' \neq s$.*
- *Let $R = (R_1, [a, b])$. Then $\mathcal{C}(R) = \mathcal{C}(R_1) \cup \{a \leq d(R_1) \leq b\}$, $d_s(R) = d_s(R_1)$ for all s, and $d(R) = d(R_1)$.*
- *Let $R = R_1{}^\frown R_2$. By renaming the variables if necessary, we can assume that $Var(\mathcal{C}(R_1)) \cap Var(\mathcal{C}(R_2)) = \emptyset$. Then, $\mathcal{C}(R) = \mathcal{C}(R_1) \cup \mathcal{C}(R_2)$, $d_s(R) = d_s(R_1) + d_s(R_2)$ and $d(R) = d(R_1) + d(R_2)$.*
- *Let $R = R_1 \otimes R_2$. Assume that $Var(\mathcal{C}(R_1)) \cap Var(\mathcal{C}(R_2)) = \emptyset$. Then, $\mathcal{C}(R) = \mathcal{C}(R_1) \cup \mathcal{C}(R_2) \cup \{d(R_1) = d(R_2)\}$, $d_s(R) = d_s(R_1) + d_s(R_2)$ for all $s \in \boldsymbol{S}$, and $d(R) = d(R_1)$.*

Let $inv(R)$ denote $\sum_{s \in \boldsymbol{S}} c_s d_s(R)$.

For instance, let R be

$$((s,[1,5])^\frown(u,[1,7]))\otimes(v,[3,10])\,.$$

Then, we can associate to each primitive in R a variable, say to $(s,[1,5])$ variable x, to $(u,[1,7])$ variable y and to $(v,[3,10])$ variable z. Then, $\mathcal{C}(R)=\{1\le x\le 5, 1\le y\le 7, 3\le z\le 10, x+y=z\}$, $d(R)=z$, $d_s(R)=x$, $d_u(R)=y$, and $d_v(R)=z$.

For a solution w of the system of linear constraints $\mathcal{C}(R)$, denote by $d_s(R)(w)$, $d(R)(w)$ and $inv(R)(w)$ respectively, the value of $d_s(R)$, $d(R)$ and $inv(R)$ evaluated over w. For vectors $w_1=(t_1,t_2,\ldots,t_p)$ and $w_2=(u_1,u_2,\ldots,u_q)$, we denote by (w_1,w_2) the vector $(t_1,t_2,\ldots,t_p,u_1,u_2,\ldots,u_q)$, and if $p=q$ we denote by w_1+w_2 the vector $(t_1+u_1,t_2+u_2,\ldots,t_p+u_p)$.

Let $\psi(R)$ denote $\max\{inv(R)\}$ subject to $\mathcal{C}(R)\cup\{A\le d(R)\le B\}$.

Theorem 15. *[34] For a simple* TRE *R, $R\models\psi$ iff $\psi(R)\le M$.*

From Theorem 15, for a simple TRE R, checking $R\models\psi$ can be done by solving the linear programming problem to find $\psi(R)$ and comparing it to M.

Example 1. Let

$$R=((s,[1,5])^\frown(u,[1,7]))\otimes(v,[3,10])$$
$$\psi=4\le\ell\le 8\Rightarrow 2\textstyle\int s-\int v\le 5q$$

Let x,y,z be variables associated to the primitives $(s,[1,5]),(u,[1,7]),(v,[3,10])$ respectively. $R\models\psi$ can be checked by solving the linear programming problem

$$\begin{aligned}\max\{2x-z\}\text{ subject to }&1\le x\le 5\\ &1\le y\le 7\\ &3\le z\le 10\\ &z=x+y\\ &4\le z\le 8\end{aligned}$$

and checking whether it is less than 5. It is easy to see that the solution of the linear programming problem is $x=5,\ y=1,\ z=6$ and the maximal value of the objective function is 4, which is less than 5. □

Let R be an infinite TRE. By replacing each occurrence of the operator $*$ (repetition) with a fresh integer variable k_i, we obtain a finite TREand can associate a finite number of linear programming problems to it. However, because the set of values of k_i's is infinite, the number of linear programming problems is also infinite. It is therefore impossible to solve all of these problems.

In the following sections, we will introduce a technique to reduce an infinite TRE to a finite TRE which is ψ-equivalent to it, and therefore an infinite TRE could be checked for ψ. This technique was first introduced by Zhou et al [73] and was generalized by us in [43] and [34].

Reducing TRE's to finite TRE's From now on we assume that any subexpression of TRE R is not of the form $(R', [0,0])$ because removing the subexpressions of that form from R does not change the set $\mathcal{M}(R)$.

Let R, R' be TRE's. If there is an occurrence of R' in R, then R' is called *sub-expression* of R. For example, let $R = ((s,[1,5])^\frown(u,[1,7])) \otimes (v,[3,10])^*$. Then $(s,[1,5])$, $(u,[1,7])$, $(v,[3,10])$, $(s,[1,5])^\frown(u,[1,7])$, $(v,[3,10])^*$ and R are sub-expressions of R. A sub-expression R' of R can occur at many different positions in R. In the sequel, when we talk about a subexpression of R, we mean an occurrence of its in R.

An TRE R for which $\mathcal{M}(R) = \emptyset$ is said to be *empty* TRE and denoted by Λ. For example, $(s,[3,5]) \otimes (v,[6,9])$ is an empty TRE. We will show how to recognise an empty expression later in the section.

For any TRE R in which there is no occurrence of an empty sub-expression, we associate with the numbers $m(R)$, $M(R)$ as follows. Roughly speaking, $m(R)$ is a lower bound and $M(R)$ is an upper bound of the set $\{d(\sigma) \mid \sigma \in \mathcal{M}(R)\}$.

Definition 7.
- *If* $R = \epsilon$, *then* $m(R) = 0$ *and* $M(R) = 0$.
- *If* $R = (R_1, [a,b])$, *then* $m(R) = \min\{m(R_1), a\}$ *and* $M(R) = \max\{M(R_1), b\}$ *(*b *may be* ∞*).*
- *If* $R = R_1^*$, *then* $m(R) = 0$ *and* $M(R) = \infty$.
- *If* $R = R_1{}^\frown R_2$, *then* $m(R) = m(R_1) + m(R_2)$ *and* $M(R) = M(R_1) + M(R_2)$.
- *If* $R = R_1 \oplus R_2$,
  then $m(R) = \min(m(R_1), m(R_2))$ *and* $M(R) = \max(M(R_1), M(R_2))$.
- *If* $R = R_1 \otimes R_2$,
  then $m(R) = \max(m(R_1), m(R_2))$ *and* $M(R) = \min(M(R_1), M(R_2))$.

From Definition 7, it is easy to see that if R is a simple TRE and $M(R) < \infty$ then $m(R), M(R)$ are minimum and maximum of the set $\{d(\sigma) \mid \sigma \in \mathcal{M}(R)\}$. Furthermore, for any TRE R, for any $\sigma \in \mathcal{M}(R)$, $m(R) \leq d(\sigma) \leq M(R)$.

For example, let

$$R = ((s,[1,5])^\frown(u,[1,7])) \otimes (v,[3,10])\,.$$

Then, $m(R) = 3$ and $M(R) = 10$. This means that for any $\sigma \in \mathcal{M}(R)$, $3 \leq d(\sigma) \leq 10$.

An important remark should be made here is that for any simple TRE R, for any real number r such that $m(R) \leq r \leq M(R)$, there is a model $\sigma \in \mathcal{M}(R)$ for which $d(\sigma) = r$. Therefore, checking the emptiness of a simple TRE R is trivial. Hence, for a simple TRE R, $m(R) = 0$ means that for any constrained expression $(R_1, [a,b])$ occurring in R the lower bound a should be 0.

Note that for any non empty TRE's R_1, R_2, the expression $R_1 \otimes R_2$ may be empty although the expressions $R_1{}^\frown R_2, R_1 \oplus R_2, R_1^*$ cannot be empty.

If R is not an empty TRE, we can find out R_1 such that R_1 has no empty sub-expression and that $\mathcal{M}(R) = \mathcal{M}(R_1)$. Thus, from now on, unless otherwise stated, we assume that all TRE's under our consideration are not empty TRE's and do not have any empty sub-expression.

Let R_1, R_2 be TRE's. As discussed earlier, if $R = R_1 \otimes R_2$ then any $\sigma \in \mathcal{M}(R)$ is constructed from models $\sigma_1 \in \mathcal{M}(R_1)$ and $\sigma_2 \in \mathcal{M}(R_2)$ such that $d(\sigma_1) = d(\sigma_2)$. Hence, the execution time of R_1 is limited by the execution time of R_2 and vice-versa. In general, the execution time of R', where R' is an arbitrary sub-expression of R_1, is not only bounded by $m(R')$ and $M(R')$ but also by $m(R_2)$ and $M(R_2)$. This means that the execution time of a sub-expression R' in a TRE R is constrained by the operator $\otimes$ and by its occurrence position in R. To capture these constraints we define the quantities $m(R', R)$ and $M(R', R)$ as lower and upper bounds of the time execution of R' when it occurs at fixed position in R. $m(R', R)$ and $M(R', R)$ are defined recursively as follows.

Definition 8.

- *Let $R = R'$. $m(R', R) = 0$ and $M(R', R) = \infty$ (no additional constraint).*
- *Let $R = R_1 \star R_2$ ($R_2 \star R_1$, R_1^*), where $\star \in \{^\frown, \oplus\}$ and R' occurs in R_1. Then $m(R', R) = m(R', R_1)$ and $M(R', R) = M(R', R_1)$ (no additional constraint).*
- *Let $R = R_1 \otimes R_2$ ($R_2 \otimes R_1$), and let R' occur in R_1. Then $m(R', R) = max(m(R', R_1), m(R_2))$ and $M(R', R) = min(M(R', R_1), M(R_2))$ (additional constraint enforced by the operator $\otimes$).*

For example, let

$$R = ((s, [1, 5])^\frown(u, [1, 7])) \otimes (v, [3, 10])^* .$$

Let $R' = (s, [1, 5])^\frown(u, [1, 7])$ then $m(R', R) = 0$ and $M(R', R) = \infty$.

Let $R' = (v, [3, 10])^*$. Then $m(R', R) = 2$ and $M(R', R) = 12$.

Denote $\mathcal{M}(R', R) = \{\sigma \in \mathcal{M}(R') \mid m(R', R) \leq d(\sigma) \leq M(R', R)\}$. From the above discussion, it can be seen that only the models in $\mathcal{M}(R', R)$ can participate in constructing models of R. Therefore, from now on, if R' is considered as a sub-expression (occurring at a fixed position) of R then we can use $\mathcal{M}(R', R)$ for $\mathcal{M}(R')$.

By induction on the structure of TRE's, we can prove the following lemmas.

Lemma 4. *Let R_1, R_2, R' be arbitrary TRE's. If for any model $\sigma_1 \in \mathcal{M}(R_1)$, there exists a model $\sigma_2 \in \mathcal{M}(R_2)$ such that $d(\sigma_1) = d(\sigma_2)$ and $inv(\sigma_1) \leq inv(\sigma_2)$ then for any model $\sigma_1' \in \mathcal{M}(R_1{}^\frown R')$ ($\mathcal{M}(R_1 \oplus R')$, $\mathcal{M}(R_1 \otimes R')$, $\mathcal{M}(R_1^*)$) there exists a model $\sigma_2' \in \mathcal{M}(R_2{}^\frown R')$ ($\mathcal{M}(R_2 \oplus R')$, $\mathcal{M}(R_2 \otimes R')$, $\mathcal{M}(R_2^*)$) such that $d(\sigma_1') = d(\sigma_2')$ and $inv(\sigma_1') \leq inv(\sigma_2')$.*

Lemma 5. *Let R, R_1, R_2 be arbitrary TRE's. If*

1. *For any model $\sigma_1 \in \mathcal{M}(R_1)$, there exists a model $\sigma_2 \in \mathcal{M}(R_2)$ such that $d(\sigma_1) = d(\sigma_2)$ and $inv(\sigma_1) \leq inv(\sigma_2)$, and*
2. *For any model $\sigma_2 \in \mathcal{M}(R_2)$, there exists a model $\sigma_1 \in \mathcal{M}(R_1)$ such that $d(\sigma_2) = d(\sigma_1)$ and $inv(\sigma_2) \leq inv(\sigma_1)$,*

then by replacing an occurrence of R_1 in R with R_2, we obtain a new expression R' which is ψ-equivalent to R, i.e. $R' \equiv_\psi R$.

Let $\lfloor x \rfloor$ be the floor of a real variable x, which is the maximal integer which are not greater than x.

Theorem 16. *[34] Let R_1 be a simple* TRE *with $m(R_1) = 0$. Let R_1' be the* TRE *obtained from R_1 by replacing each subexpression of the form $(R'', [0, b])$ of R_1 with $(R'', [0, \infty))$ (remember that $b > 0$ as assumed earlier). Then, by replacing an occurrence of R_1^* in a* TRE *R with R_1', we obtain a new expression R' which is ψ-equivalent to R.*

Theorem 17. *[34] Let R_1 be a simple* TRE *with $m(R_1) > 0$. Let R_1^* be an occurrence of the* TRE *R_1^* in a* TRE *R for which $M(R_1^*, R) < \infty$ or $B < \infty$ (recall that B is the upper bound of the observation time period in the premise $A \leq \ell \leq B$ of the linear duration invariant ψ). Let $R_1' = \oplus_{i=0}^{k} R_1^i$, where $k = \lfloor min\ \{M(R_1^*, R), B\}/(m(R_1)\rfloor + 1)$. Then by replacing the occurrence R_1^* in R with R_1', we obtain a new expression R' which is ψ-equivalent to R.*

Let R' be a sub-expression of R. R' is said to be under $\otimes$ if there is a sub-expression of form $R_1 \otimes R_2$ of R such that R' occurs either in R_1 or in R_2. If A^* is not under $\otimes$, then by definition 8, $M(A^*, R) = \infty$.

Given a simple TRE R_1. Let $maxinv(R_1)$ denote the maximal value of $\{inv(\sigma) \mid \sigma \in \mathcal{M}(R_1)\}$. $maxinv(R_1)$ can be computed by solving the linear programming problem: finding the maximum of the objective function $\sum_{s \in \mathbf{S}} c_s d_s(R_1)$ subject to the set of constraints $\mathcal{C}(R_1)$.

Lemma 6. *Let K be a real number, R_1^* be a sub-expression of R which is not under $\otimes$, where R_1 is a simple* TRE *with $m(R_1) > 0, maxinv(R_1) \leq 0$. Assume that $B = \infty$. Furthermore, let R' be obtained from R by replacing the occurrence R_1^* in R by $R_1' = \bigoplus_{i=0}^{k} R_1^i$ with $k = \lfloor K/m(R_1) \rfloor + 1$.*

Then, for any model $\sigma \in \mathcal{M}(R)$ such that $d(\sigma) \geq K$, there exists a model $\sigma' \in \mathcal{M}(R')$ such that $d(\sigma') \geq K$, and $inv(\sigma) \leq inv(\sigma')$.

Theorem 18. *[34] Let $B = \infty$, and R_1^* be a sub-expression of R such that R_1^* is not under $\otimes$, where R_1 is a simple* TRE *with $m(R_1) > 0$. Then*

1. *If $maxinv(R_1) \leq 0$, then by replacing R_1^* in R with $R_1' = \oplus_{i=0}^{k} R_1^i$, where $k = \lfloor A/m(R_1) \rfloor + 1$, we obtain a new expression R' such that $R' \equiv_\psi R$.*
2. *If $maxinv(R_1) > 0$, then $R \not\models \psi$.*

Proof.

1. It is obviously that $\mathcal{M}(R_1') \subset \mathcal{M}(R_1^*)$. Hence, by Lemma 4, for any model $\sigma' \in \mathcal{M}(R')$, there exists model $\sigma \in \mathcal{M}(R)$ such that $d(\sigma) = d(\sigma')$ and $inv(\sigma) \leq inv(\sigma')$. By Lemmas 4 and 5, it follows that $R \models \psi$ implies $R' \models \psi$. The other direction is proved as follows. By lemma 6, for any $\sigma \in \mathcal{M}(R)$ such that $d(\sigma) \geq A$ there is $\sigma' \in \mathcal{M}(R')$ such that $d(\sigma') \geq A$ and $inv(\sigma') \geq inv(\sigma)$. Hence, as a result, $R' \models \psi$ implies $R \models \psi$.

2. Assume that $maxinv(R_1) > 0$ and R_1^* is not under $\otimes$. By induction on the structure of R, it can be seen easily that for any subexpression R_1 of R if there exists a sequence of models $\sigma_i \in \mathcal{M}(R_1)$, $i \geq 1$ such that $\lim inv(\sigma_i) = \infty$, then there exists a sequence of models $\sigma_i' \in \mathcal{M}(R)$, $i \geq 1$ such that $\lim_{i\to\infty} inv(\sigma_i') = \infty$. Let w_0 be the optimal solution of the linear programming problem: $\max \sum_{s\in\mathbf{S}} c_s d_s(R_1)$ subject to $\mathcal{C}(R_1)$. Let $\sigma_0 \in \mathcal{M}(R_1)$ be the model corresponding to w_0 then $inv(\sigma_0) = maxinv(R_1) > 0$. Hence, for the sequence $\sigma_i = \sigma_0{}^\frown\sigma_0{}^\frown\ldots{}^\frown\sigma_0$ (i times), $i \geq 1$, we have for all i, $\sigma_i \in \mathcal{M}(R_1^*)$ and $inv(\sigma_i) = i \times inv(\sigma_0) \to \infty$ when $i \to \infty$. Since R_1^* is a sub-expression of R, we can construct a sequence of models σ_i', $i \geq 1$ such that $\sigma_i' \in \mathcal{M}(R)$ and $inv(\sigma_i') \to \infty$ (when $i \to \infty$). Hence, we can find a model $\sigma \in R$ satisfying that $A \leq d(\sigma) \leq \infty$ and $inv(\sigma) > M$. In the other words, $R \not\models \psi$.

□

By Theorems 16, 17 and 18, we can remove a star in a TRE R without introducing a new star, without increasing the number of stars under $\otimes$ for the following cases:

- a star of the form R_1^*, where R_1 is a simple TRE with $m(R_1) = 0$ (Theorem 16),
- a star of the form R_1^*, where R_1 is a simple TRE with $m(R_1) > 0$ for the case that either $M(R_1^*, R)$ or B is finite (Theorem 17),
- a star of the form R_1^*, where R_1 is a simple TRE with $m(R_1) > 0$ for the case that B is infinite and R_1^* is not under $\otimes$ (Theorem 18).

Therefore, if the linear duration ψ is of the form $A \leq \ell \leq B \Rightarrow \sum_{s\in\mathbf{S}} c_s \int s \leq M$ for which $B < \infty$ or if the TRE R has no sub-expression of the form R_1^*, where R_1 is a simple TRE with $m(R_1) > 0$ and $M(R_1^*, R) = \infty$, then checking $R \models \psi$ can be reduced to solving a finite number of linear programming problems.

As said earlier, what we still need to do in using this technique is to check the emptiness of TRE's. Checking the emptiness of a TRE in which the star does not occur under $\otimes$ is so trivial. However, the problem becomes difficult when the star occurs in the operands of a $\otimes$. Let for example

$$R = ((s_1, [a_1, b_1])^*{}^\frown(s_2, [a_2, b_2])^*)\otimes \\ ((s_3, [a_3, b_3])^*{}^\frown(s_4, [a_4, b_4])^*)$$

Replacing each star $*$ with an integral variable, we get

$$R = \bigoplus_{m_1,m_2,m_3,m_4\geq 0} \begin{pmatrix} ((s_1, [a_1, b_1])^{m_1}{}^\frown(s_2, [a_2, b_2])^{m_2}) \\ \otimes \\ ((s_3, [a_3, b_3])^{m_3}{}^\frown(s_4, [a_4, b_4])^{m_4}) \end{pmatrix}.$$

Thus, R is not empty iff the inequalities

$$m_1a_1 + m_2a_2 \leq m_3b_b + m_4b_4 \\ m_3a_3 + m_4a_4 \leq m_1b_1 + m_2b_2 \\ m_1 \geq 0, m_2 \geq 0, m_3 \geq 0, m_4 \geq 0$$

has an integral solution. In order to make the problem easier, we assume in this section that all the real constants occurring in a TRE are rational. Thus, checking the emptiness of TRE's leads to checking the emptiness of $\{\mathbf{Ax} \leq \mathbf{b} | \mathbf{x} \text{ integral}\}$, where $\mathbf{A}$ is a rational matrix, $\mathbf{b}$ is a rational vector, which is an integer linear programming problem and can be solved in polynomial time.

For the TRE's in which there are occurrences of the operator $*$ under $\otimes$ and $B = \infty$ in the LDI D, the problem is difficult, and we have to use mixed integer linear programming techniques ([39, 34]) to solve the problem.

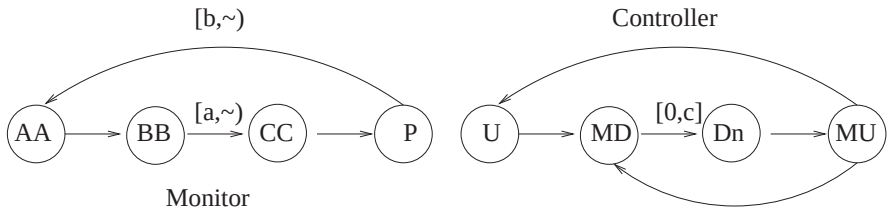


Fig. 5. A Railroad Crossing Monitor. Transitions from AA to BB and from U to MD are synchronised, and transitions from CC to P and from Dn to MU are synchronised.

Example 2. Let us take the railroad crossing system [66] as an example to illustrate the checking technique. We have trains, a railroad crossing monitor, and a gate controller which are subject to the following constraints (see Figure 5).

1. The monitor has four states to express the positions of train: state AA for train approaching beyond 1/2 mile, state BB for train approaching within 1/2 mile, state CC for train crossing, and state P for train just passed.
2. The controller has four states to express the positions of the gate: state U for the gate being up, state MD for the gate moving down, state Dn for the gate being down and MU for the gate moving up.

When the system starts, the monitor is in state AA and the controller is in state U. In state AA, when the monitor detects that a train approaching within 1/2 mile, it enters state BB, and at the same time if the controller is in state U or state MU, it must enters state MD. Namely, when the gate is up or is moving up, and detects that the monitor enters state BB, it must start moving down. When the train enters the crossing, the monitor enters state CC, and when the train has passed, it enters state P. When the monitor changes its state from CC to P then at the same time the monitor changes its state from Dn to MU. This means, when the gate is down, and detects that the monitor enters state P, it begins to move up. In addition, due to the speed of trains and the safety distance between trains, it takes at least a time units for a train to go to the crossing, after entering state BB, and when a train has passed, a new train could come after at least b time units. That means that the monitor stays in BB at least a time units each time and in P at least b time units each time. Furthermore, assume that it

takes the gate at most c time units to move down, and hence, the controller stays at MD at most c time units each time, where $c \leq a$. The automata modelling the railroad crossing system are depicted in Figure 5. Intuitively, the parallel behaviour of the system is now can be described by the following TRE RCM:

$$
\begin{array}{l}
(AA \otimes U)^\frown \\
(((BB, [a, \infty))^\frown CC \otimes (MD, [0, c])^\frown Dn)^\frown \\
((P, [b, \infty))^\frown AA \otimes (MU^\frown U \oplus MU)))^* \frown \\
(\epsilon \oplus \\
(BB \oplus (BB, [a, \infty))^\frown CC) \otimes ((MD, [0, c]) \oplus (MD, [0, c])^\frown Dn) \oplus \\
((BB, [a, \infty))^\frown CC \otimes (MD, [0, c])^\frown Dn)^\frown \\
((P \oplus P^\frown AA) \otimes (MU \oplus MU^\frown U)))
\end{array}
$$

Now we verify that the railroad crossing monitor satisfies the requirement for the system. That is to check $RCM \models D$, where D is $0 \leq \ell \leq \infty \Rightarrow \int CC - \int Dn \leq 0$.

Because the subexpression under $*$ is not a simple one, in order to use Theorem 18, we transform RCM into the following expression $RCM1$ using Theorem 14:

$$
\begin{array}{rl}
RCM1 \equiv & (AA \otimes U)^\frown \\
& (((BB, [a, \infty))^\frown CC \otimes (MD, [0, c])^\frown Dn)^\frown \\
& ((P, [b, \infty))^\frown AA \otimes MU^\frown U))^* \frown \\
& (((BB, [a, \infty))^\frown CC \otimes (MD, [0, c])^\frown Dn)^\frown \\
& ((P, [b, \infty))^\frown AA \otimes MU))^* \frown \\
& (\epsilon \oplus \\
& (BB \oplus (BB, [a, \infty))^\frown CC) \otimes \\
& \quad ((MD, [0, c]) \oplus (MD, [0, c])^\frown Dn) \oplus \\
& ((BB, [a, \infty))^\frown CC \otimes (MD, [0, c])^\frown Dn)^\frown \\
& ((P \oplus P^\frown AA) \otimes (MU \oplus MU^\frown U)))
\end{array}
$$

Let

$$
\begin{array}{rl}
R_1 = & (((BB, [a, \infty))^\frown CC \otimes (MD, [0, c])^\frown Dn)^\frown \\
& ((P, [b, \infty))^\frown AA \otimes MU^\frown U)) \\
R_2 = & (((BB, [a, \infty))^\frown CC \otimes (MD, [0, c])^\frown Dn)^\frown \\
& ((P, [b, \infty))^\frown AA \otimes MU))
\end{array}
$$

By Definition 7, $m(R_1) > 0$ and $m(R_2) > 0$. Furthermore,

$$
\begin{array}{rl}
maxinv(R_1) = & \max\{inv(\sigma) | \sigma \in \mathcal{M}(R_1)\} \\
= & \max\{t_2 - t_4\} \text{ subject to} \\
& a \leq t_1,\ 0 \leq t_2 \\
& 0 \leq t_3 \leq c,\ 0 \leq t_4 \\
& t_1 + t_2 = t_3 + t_4,\ b \leq t_5 \\
& 0 \leq t_6,\ 0 \leq t_7 \\
& 0 \leq t_8,\ t_5 + t_6 = t_7 + t_8 \\
= & c - a
\end{array}
$$

Similarly, we have $maxinv(R_2) = c - a$ as well. Since $c \leq a$, we have $maxinv(R_1) \leq 0$ and $maxinv(R_2) \leq 0$. By applying Theorem 18 twice with noticing that $k = 1$, we have that $RCM \models D$ is now equivalent to $RCM2 \models D$, where

$$
\begin{aligned}
RCM2 = {} & (AA \otimes U)^\frown \\
& (\epsilon \oplus \\
& (((BB, [a, \infty))^\frown CC \otimes (MD, [0, c])^\frown Dn)^\frown \\
& ((P, [b, \infty))^\frown AA \otimes MU^\frown U)))^\frown \\
& (\epsilon \oplus \\
& (((BB, [a, \infty))^\frown CC \otimes (MD, [0, c])^\frown Dn)^\frown \\
& ((P, [b, \infty))^\frown AA \otimes MU))) ^\frown \\
& (\epsilon \oplus \\
& (BB \oplus (BB, [a, \infty))^\frown CC) \otimes ((MD, [0, c]) \oplus \\
& (MD, [0, c])^\frown Dn) \oplus \\
& (BB^\frown CC \otimes (MD, [0, c])^\frown Dn)^\frown \\
& ((P \oplus P^\frown AA) \otimes (MU \oplus MU^\frown U)))
\end{aligned}
$$

$RCM2$ is a finite TRE, and checking $RCM2 \models D$ is so simple for this case using Theorem 15. □

For the class of real time systems whose behaviours are described by general timed automata, if the constants occurring in the linear Duration Invariant are integer, then checking can also be done by investigating the region graph of the timed automaton modelling the system. Readers are referred to [38, 65] for the details of the technique.

8 A Case Study: Modeling and Verification of the Bi-phase Mark Protocol in Duration Calculus

It is natural and convenient to model time as non-negative real numbers. However, during the development of a real-time computing system, we may have to use the set of natural numbers for time. This is specially true when dealing with the implementation of the system in computer which running in discrete time. In this section, we introduce some techniques for using Duration Calculus in modelling and specification of some discrete time structures. These techniques demonstrate how to use temporal propositional letters with specific meanings in additional to state variables in modelling and specification. We then use these techniques for modelling and verifying the correctness of the well-known case study "Biphase Mark Protocol".

For our convenience, we extend DC with the formula $\lceil P \rceil^0$ that has been introduced by Pandya in his early work [49]. The semantic for $\lceil P \rceil^0$ is defined by $\mathcal{I}, [a, b] \models \lceil P \rceil^0$ iff $a = b$ and $P_{\mathcal{I}}(a) = 1$.

8.1 Discrete Duration Calculus Models

Recall that discrete models of Duration Calculus use the set of natural numbers $\mathbb{N}$, which is a subset of $\mathbb{R}^+$, for time (we assume that $0 \in \mathbb{N}$). We can embed

the discrete time models into continuous time models by considering a state variable in discrete DC models as a state in continuous models that can change its value only at the integer points. For that purpose, we introduce several fresh temporal propositional letters and state variables with specific meaning. Let *int* be a temporal propositional letter with the meaning that *int* is interpreted as 1 for an interval if and only if the interval is from an integer to an integer, i.e. for any interpretation $\mathcal{I}$, $int_{\mathcal{I}}([a,b]) = 1$ iff $a, b \in \mathbb{N}$. The axioms to characterise the properties of the temporal propositional letter *int* can be given as follows. First, the integer intervals have integral endpoints, and remain integer intervals when extended by 1 time unit:

$$int \Rightarrow ((int \wedge \ell = 0)^\frown(int \wedge \ell = 1)^*) \wedge ((int \wedge \ell = 1)^*{}^\frown(int \wedge \ell = 0)) \quad (6)$$

$$int^\frown(\ell = 1) \Rightarrow int \quad (7)$$

Second, $int \wedge \ell = 1$ should be a unique partition of the greatest integral subinterval of any interval with length 2 or more, i.e.

$$\begin{aligned} \ell \geq 2 \Rightarrow \ell < 1^\frown((int \wedge \ell = 1)^* \wedge \\ \neg(true^\frown(int \wedge \ell = 1)^\frown\neg(int \wedge \ell = 1)^*) \wedge \\ \neg(\neg(int \wedge \ell = 1)^*{}^\frown(int \wedge \ell = 1)^\frown true))^\frown \\ \ell < 1 \end{aligned} \quad (8)$$

Similarly to Lemma 3.2 in [20] we can show that the axiom 8 is equivalent to the fact that any interval $[b, e]$ that have the length 2 or longer has the unique set of time points $b \leq \tau_0 < \tau_1 < \ldots < \tau_m \leq e$ such that $\mathcal{I}, [\tau_i, \tau_{i+1}] \models int \wedge \ell = 1$, $\tau_0 - b < 1$ and $e - \tau_m < 1$, and $[\tau_i, \tau_{i+1}]$ are the only subintervals of $[b, e]$ that that satisfy $(int \wedge \ell = 1)$.

Let $\mathcal{ID}$ denote the set of these three axioms 6, 7 and 8. $\mathcal{ID}$ specifies all the properties of integer intervals except that their endpoints are integer as formulated by:

Proposition 2. *[31]*

1. *Let $\mathcal{I}$ be an interpretation satisfying that $int_{\mathcal{I}}([b,e]) = true$ iff $[b,e]$ is an integer interval. Then $\mathcal{I}, [b,e] \models D$ for any integer interval $[b.e]$, and for any formula $D \in \mathcal{ID}$.*
2. *Let $\mathcal{I}$ be an interpretation satisfying that $\mathcal{I}, [b,e] \models D$ for any interval $[b.e]$, and for any formula $D \in \mathcal{ID}$. Then, $int_{\mathcal{I}}([0,0]) = true$ implies that for $int_{\mathcal{I}}([b,e]) = true$ iff $[b,e]$ is an integer interval.*

Proof. The item 1 is obvious, and we only give a proof of Item 2 here. Let us consider an interval $[0, n]$ with $n > 100$. From the fact that $\mathcal{I}, [0, n] \models D$ where D is the formula 8, we have that there are points $0 \leq \tau_0 < \tau_1 < \ldots < \tau_m \leq e$ such that $\mathcal{I}, [\tau_i, \tau_{i+i}] \models int \wedge \ell = 1$, $\tau_0 < 1$ and $n - \tau_m < 1$, and

$$\begin{aligned} \mathcal{I}, [\tau_0, \tau_m] \models (\neg(true^\frown(int \wedge \ell = 1)^\frown\neg(int \wedge \ell = 1)^*) \wedge \\ \neg(\neg(int \wedge \ell = 1)^*{}^\frown(int \wedge \ell = 1)^\frown true)) \end{aligned}$$

If $\tau_0 > 0$, from the axiom 7, it follows that $\mathcal{I}, [0,k] \models int$ for all $k \in \mathbb{N}$ and $k \leq n$ and $k < \tau_k < k+1$. Applying the axiom 6 for the interval $[0,k]$ implies that $\mathcal{I}, [k,k+1] \models int \wedge \ell = 1$. Consequently, $\mathcal{I}, [m-1, \tau_m] \models \neg(int \wedge \ell = 1)^*$ and $\mathcal{I}, [m-2, m_1] \models (int \wedge \ell = 1)$. This is a contradiction to $\mathcal{I}, [\tau_0, \tau_m] \models \neg(true^\frown(int \wedge \ell = 1)^\frown\neg(int \wedge \ell = 1)^*)$. □

Note that Item 2 of Proposition 2 can be generalised as

> Let $\mathcal{I}$ be an interpretation satisfying that $\mathcal{I}, [b,e] \models D$ for any interval $[b.e]$, and for any formula $D \in \mathcal{ID}$. Let $h \in \mathcal{R}^+$, $h < 1$. Then, $int_\mathcal{I}([h,h]) = true$ implies that for $int_\mathcal{I}([b,e]) = true$ iff $[b,e]$ is of the form $[h+n, h+m]$, $m, n \in \mathbb{N}$ and $n \leq m$.

So, $\mathcal{ID}$ is a set of formulas specifying the set of intervals of a discrete time obtained by shifting $\mathbb{N}$ by h time units ($h < 1$).

Another way to express integer intervals is to use a state variable C be that changes its value at each natural number which represents a tick of the real-time clock, i.e. $C_\mathcal{I}(t) = 1$ iff $\lfloor t \rfloor$ is odd.

The state variable C can also express if an interval is an integer interval. Namely, we have

$$
\begin{array}{l}
(\lceil C \rceil \vee \lceil \neg C \rceil) \wedge \ell = 1 \Rightarrow int \\
int \wedge \ell = 1 \Rightarrow (\lceil C \rceil \vee \lceil \neg C \rceil) \\
\lceil C \rceil^\frown\lceil \neg C \rceil \Rightarrow true^\frown int^\frown true \\
\lceil \neg C \rceil^\frown\lceil C \rceil \Rightarrow true^\frown int^\frown true \\
(int \wedge \ell = 1)^\frown(int \wedge \ell = 1) \Rightarrow \\
\quad ((\lceil C \rceil \wedge \ell = 1)^\frown(\lceil \neg C \rceil \wedge \ell = 1)) \vee \\
\quad\quad ((\lceil \neg C \rceil \wedge \ell = 1)^\frown(\lceil C \rceil \wedge \ell = 1))
\end{array}
$$

Let $\mathcal{CC}$ denote the set of these formulas. $\mathcal{CC}$ specifies all the properties of the special clock state variable C. Any interval satisfying $int \wedge \ell > 0$ can be expressed precisely via a DC formula with state variable C (without int). Perhaps $int \wedge \ell = 0$ is the only formula that cannot be expressed by a formula via state variable C without int. $\mathcal{CC}$ can also be used as a means to define the variable C via int and vice-versa. If we use $\mathcal{CC}$ to define int, the axioms for C simply are:

$$\lceil C \rceil \vee \lceil \neg C \rceil \Rightarrow \ell \leq 1 \tag{9}$$

$$((\lceil C \rceil^\frown\lceil \neg C \rceil^\frown\lceil C \rceil) \vee (\lceil \neg C \rceil^\frown\lceil C \rceil^\frown\lceil \neg C \rceil)) \Rightarrow \ell \geq 1 \tag{10}$$

The relationship between these axioms and the axioms for int presented earlier is formulated as:

Proposition 3. *[31] Let interpretation $\mathcal{I}$ be such that the formulas in $\mathcal{CC}$ and axioms (6) and (7) are satisfied by all intervals.*

1. *If the axioms (9) and (10) are satisfied by all intervals, the axiom (8) is satisfied by all intervals.*
2. *If the axiom (8) is satisfied by all intervals then the axioms (9) and (10) are satisfied by all intervals, too.*

Proof.

Proof of Item 1. The axioms (9) and (10) implies that the formula

$$(\lceil\neg P\rceil^\frown\lceil P\rceil^\frown\lceil\neg P\rceil) \Rightarrow (\lceil\neg P\rceil^\frown(\lceil P\rceil \wedge \ell = 1)^\frown\lceil\neg P\rceil)$$

is satisfied for any interval when P is either C or $\neg C$. For any interval $[b, e]$, if $e - b \geq 2$ then there are $b = \tau_0 < \ldots < \tau_n = e$ such that $[\tau_i, \tau_{i+1}]$ satisfies $\lceil C\rceil \vee \lceil\neg C\rceil$, and τ_i, $0 < i < n$ are the points the state C changes its value. Therefore, from (3), (4) and $\mathcal{CC}$ the formula $int \wedge \ell = 1$ is satisfied by $[\tau_i, \tau_{i+1}]$ when $0 < i < n - 1$, and $\tau_1 - \tau_0 < 1$ and $\tau_n - \tau_{n-1} < 1$. Furthermore, from $int \wedge \ell = 1 \Rightarrow (\lceil C\rceil \vee \lceil\neg C\rceil)$ it follows that $[\tau_i, \tau_{i+1}]$, $0 < i < n - 1$ are the only intervals satisfying $int \wedge \ell = 1$. Hence, (8) is satisfied by $[b, e]$.

Proof of Item 2. Let $h > 0$ be the first time point that state C changes its value. From the axioms (6), (7) and (8) it follows that $h \leq 1$ and $int \wedge \ell = 1$ is satisfied by and only by the intervals of the form $[n+h, n+1+h]$, $n \in \mathbb{N}$. Hence, if $\mathcal{CC}$ is satisfied by all intervals, the axioms (9) and (10) are also satisfied by all intervals. □

So, with the assumption that 0 is an integer point, the axioms (9) and (10) are equivalent to the axiom (8).

Let *step* be a temporal propositional letter that represents two consecutive state changes of the system under consideration. When there are several state changes at a time point t, *step* evaluates to 1 over interval $[t, t]$. When two consecutive state changes are at t and t' such that $t \neq t'$, *step* is true for the interval $[t, t']$, and for any state variable P, either $\lceil P\rceil$ or $\lceil\neg P\rceil$ holds for the interval $[t, t']$. This is represented by:

$$step \wedge \ell > 0 \Rightarrow (\lceil P\rceil \vee \lceil\neg P\rceil) \text{ for any state variable } P$$
$$step \wedge \ell > 0 \Rightarrow \neg((step \wedge \ell > 0)^\frown(step \wedge \ell > 0))$$

Let $\mathcal{SC}$ denote this class of formulas.

Now consider two kinds of Duration Calculus semantics which are different from the original one defined earlier for continuous time, and called discrete semantics and discrete step time semantics.

Discrete Duration Calculus semantics are defined in the same way as for continuous time semantics except that all intervals are integer intervals. So, a, b, m and m_i in the definition should be integers instead of reals, and an interpretation $\mathcal{I}$ should assign to each state variable P a function from $\mathbb{N}$ to $\{0, 1\}$, and then expanded to a function from $\mathcal{R}^+$ to $\{0, 1\}$ by letting $\mathcal{I}_P(t) = \mathcal{I}_P(\lfloor t\rfloor)$ which is right continuous, and could be discontinuous only at integer time points. Let us use $\models_{DDC}$ to denote the modelling relation in these semantics.

Similarly, discrete step time Duration Calculus semantics are defined by restricting the set of intervals to that of intervals between state change time points. So, a, b, m and m_i in the definition should be time points where states change, and an interpretation $\mathcal{I}$ should assign to each state variable P a function from $\mathcal{S}$ to $\{0, 1\}$, where $\mathcal{S}$ is a countable subset of $\mathcal{R}^+$ intended to be the set of time points for state changes that includes the set $\mathbb{N}$. $\mathcal{I}_P$ is then expanded to

a function from $\mathcal{R}^+$ to $\{0,1\}$ by letting $\mathcal{I}_P(t) = \mathcal{I}_P(t_s)$, where $t \in \mathcal{R}^+$ and $t_s = \max\{t' \in \mathcal{S} \mid t' \leq t\}$. Then $\mathcal{I}_P(t)$ is also right continuous, and could be discontinuous only at a point in $\mathcal{S}$. Let us use $\models_{SDC}$ to denote the modelling relation in this semantics.

To express that states are interpreted as right continuous functions, we can use formula called $\mathcal{RC}$

$$[\![P]\!] \Rightarrow [\![P]\!]^0 \frown [\![P]\!] \text{ for any state variable } P$$

In [48], Pandya also proposed a semantics using only the intervals of the form $[0,t]$. This semantics is often used in model checking when only the properties in the intervals of that form is specified, and we have to check if an automata model of the system satisfied those properties during its life. We can also specify this interval model with a temporal propositional letter Pre. Pre is interpreted as true only for the interval of the form $[0,t]$. Pre is specified by the set of formulas $Pref$ defined as

$$\begin{array}{l} Pre \frown true \Rightarrow Pre \\ \neg(\ell > 0 \frown Pre) \\ Pre \wedge D \Rightarrow (Pre \wedge \ell = 0) \frown D \\ Pre \wedge (D1 \frown D2) \Rightarrow (Pre \wedge D1) \frown D2 \end{array}$$

Proposition 4. *[31] Let $\mathcal{I}$ be an interpretation that validates the set of formulas $Pref$ and $\mathcal{I}, [0,0] \models Pre$. Then, $\mathcal{I}, \mathcal{V}, [a,b] \models Pre$ iff $a = 0$.*

Proof. Straightforward □

Then, a formula D is valid in the prefix time interval model if and only if $Pre \Rightarrow D$ is a valid formula in the original model of time interval.

So far, we have introduced special temporal propositional letters *int*, *step* and Pre together with DC formulas specifying their special features. We are going to show that with these propositional letters we can provide a complete description of many useful time models.

Integer Time Model To specify that a state can only change at an integer time point, we can use the formula $\mathcal{IS}$:

$$step \Rightarrow int$$

Let $\mathcal{DL}$ be the union of $\mathcal{SC}$, $\mathcal{IS}$, $\mathcal{ID}$, $\mathcal{RC}$. $\mathcal{DL}$ forms a relative complete specification for the discrete time structure. Let φ be a formula which does not have any occurrence of temporal variables *int* ans *step*. Let $intemb(\varphi)$ be a formula that obtained from φ by replacing each proper subformula ψ of φ by $\psi \wedge int$. For example $intemb(\phi \frown \neg\psi) = (\phi \wedge int) \frown (int \wedge \neg(\psi \wedge int))$.

Theorem 19. *[31] Let φ be a DC formula with no occurrence of temporal proposition letters. Then, $\mathcal{DL} \vdash int \Rightarrow intemb(\varphi)$ exactly when $\models_{DDC} \varphi$.*

Proof. Any discrete time model $\mathcal{I}, [a, b]$ can be extended to a model that satisfies the formulas in $\mathcal{DL}$ in the obvious way, namely with the interpretation for *int* and *step* with the intended meanings for them. By induction on the structure of the formula φ, it is easy to prove that $\mathcal{I}, [a, b] \models_{DDC} \varphi$ if and only $\mathcal{I}, [a, b] \models$ $intemb(\varphi)$.

Then, the "only if" part follows directly from the soundness of the proof of the DC system that $intemb(\varphi)$ is satisfied by any integer model that satisfies $\mathcal{DL}$.

The "if" part is proved as follows. From the above observation, if $\models_{DDC}$ φ then $int \Rightarrow intemb(\varphi)$ is a valid formula in DC with the assumption $\mathcal{DL}$. Consequently, from the relative completeness of DC, $intemb(\varphi)$ is provable in DC with the assumption $\mathcal{DL}$. □

Discrete Step Time Model As it was said earlier, a discrete step time model consists of all time points at which there is a the state change. Since we have assumed that the special state variable C for the clock ticks is present in our system that changes its value at every integer point, this model of time should also include the set of natural numbers. This is the reason that we include $\mathbb{N}$ as a subset of $\mathcal{S}$. This time model was defined and used by Pandya et al in [48]. To represent a time point in this model, we introduce a temporal propositional letter *pt*, *pt* holds for an interval $[t, t']$ iff $t = t'$ and t is a time point at which there is a state change. *pt* should satisfy:

$$
\begin{aligned}
&pt \Rightarrow \ell = 0\\
&step \Rightarrow pt^\frown true^\frown pt\\
&int \Rightarrow pt^\frown true^\frown pt\\
&int \Rightarrow pt^\frown step^*
\end{aligned}
$$

Let $\mathcal{DP}$ denote this set of formulas. The last formula in this set expresses our assumption that no Zeno computation is allowed, i.e. in any time interval, there are only a finite number of state changes. Let us define a DC formula *dis* as

$$dis \mathrel{\widehat{=}} (pt^\frown true^\frown pt)$$

dis represents an interval between two discrete points. When considering the Discrete Step Time Models, the chop point should satisfy *pt*.

The sublanguage $\mathcal{DSL}$, which is the union of $\mathcal{SC}$, $\mathcal{ID}$, $\mathcal{CC}$, $\mathcal{DP}$ $\mathcal{DC}$ and $\mathcal{RC}$, forms a relatively complete specification for the discrete time structure.

Let $disemb(\varphi)$ be a formula that is obtained from φ by replacing each proper subformula ψ of φ by $\psi \wedge dis$. For example $disemb(\phi^\frown\neg\psi) = (\phi \wedge dis)^\frown(dis \wedge \neg(\psi \wedge dis)$.

Theorem 20. *[31] Let φ be a DC formula with no occurrence of temporal proposition letters. Then, $\mathcal{DSL} \vdash dis \Rightarrow disemb(\varphi)$ exactly $\models_{SDC} \varphi$.*

Proof. The proof works in exactly the same way as the proof of Theorem 19.

Any discrete step time model $\mathcal{I}, [a, b]$ can be extended to a model that satisfies formulas in $\mathcal{DL}$ in the obvious way, namely with the interpretation for *int* and

step with the intended meanings for them. By induction on the structure of the formula φ, it is easy to prove that $\mathcal{I}, [a, b] \models_{SDC} \varphi$ if and only $\mathcal{I}, [a, b] \models intemb(\varphi)$.

Then, the "only if" part follows directly from the soundness of the proof of the DC system that $intemb(\varphi)$ is satisfied by any discrete step time model that satisfies $\mathcal{DL}$.

For the "if" part, notice that if $\models_{SDC} \varphi$ then $dis \Rightarrow intemb(\varphi)$ is a valid formula in DC with the assumption $\mathcal{DL}$. Consequently, from the relative completeness of DC, $disemb(\varphi)$ is provable in DC with the assumption $\mathcal{DL}$. □

Sampling Time Models A sampling time model consists of the time points where we sample the data. Assume that the samplings are frequent enough and that any state change should be at a sampling point. To specify this time model, we can use $\mathcal{DSL}$ and an additional assumption

$$step \Rightarrow \ell = 1/h$$

where $h \in \mathbb{N}$, $h > 0$, i.e. $1/h$ is the sampling time step. Let $\mathcal{SL}_h$ be the language for the sampling time model with the sampling time step $1/h$.

8.2 Specifying Sampling, Periodic Task Systems

Sampling Sampling and specifying periodic task systems are immediate applications of the results presented in the previous section.

We have built a language for sampling time models based on the continuous time DC. Hence, we can use the proof system of DC to reason about validity of a formula in that time and state model. How to relate the validity of a formula D in that time and state model with the validity of a formula D' in the original DC? In our early work [35], we have considered that relation, but had to formulate the results in a natural meta language due to the use of different semantic models. With the help from the time modeling language, we can also formulate the relationship as formulas in DC.

Let P be a state variable. Let P_h be a state in the sampling time model with the sampling time step $1/h$ such that P_h is interpreted the same as P at any sampling time point, i.e. $\Box(pt \Rightarrow (\lceil P \rceil^0 \Leftrightarrow \lceil P_h \rceil^0)$ (denoted by $samp(P, P_h)$), and $\Box(step \wedge \ell > 0 \Rightarrow (\lceil P_h \rceil \vee \lceil \neg P_h \rceil))$ (denoted by $dig(P_h)$). Let $stable(P, d)$ denote the formula $\Box((\lceil \neg P \rceil^\frown \lceil P \rceil^\frown \lceil \neg P \rceil) \Rightarrow \ell \geq d)$.

Theorem 21. *Let $d > 1/h$. The following formulas are valid in DC:*

1. $(stable(P, d) \wedge samp(P, P_h) \wedge dig(P_h)) \Rightarrow$
   $(\int P = m \Rightarrow |\int P_h - m| \leq \min\{\ell, (\ell/d + 1)1/h\}$
2. $(stable(P, d) \wedge samp(P, P_h) \wedge dig(P_h)) \Rightarrow$
   $(\int P = m \wedge dis) \Rightarrow |\int P_h - m| \leq \min\{\ell, 1/h\ell/d\}$
3. $(stable(P, d) \wedge samp(P, P_h) \wedge dig(P_h)) \Rightarrow$
   $\int P_h = m \Rightarrow |\int P - m| \leq (\ell/d + 1)1/h$
4. $(stable(P, d) \wedge samp(P, P_h) \wedge dig(P_h)) \Rightarrow$
   $\int P_h < m \Rightarrow \int P < m + 1/h(\ell/d + 1)$

5. $(stable(P,d) \wedge samp(P,P_h) \wedge dig(P_h)) \Rightarrow$
   $\int P < m \Rightarrow \int P_h < m + 1/h(\ell/d+1)$
6. $(stable(P,d) \wedge samp(P,P_h) \wedge dig(P_h)) \Rightarrow$
   $\int P_h > m \Rightarrow \int P > m - 1/h(\ell/d+1)$
7. $(stable(P,d) \wedge samp(P,P_h) \wedge dig(P_h)) \Rightarrow$
   $\int P > m \Rightarrow \int P_h > m - 1/h(\ell/d+1)$
8. $(stable(P,d) \wedge samp(P,P_h) \wedge dig(P_h)) \Rightarrow$
   $dis \Rightarrow (\lceil\!\lceil P_h \rceil\!\rceil \Leftrightarrow \lceil\!\lceil P \rceil\!\rceil)$

Proof. This is just a reformulation of Theorem 1 in [35]. □

This theorem is useful for deriving a valid formula in the original DC from valid formulas in discrete time model. It can be used in approximate reasoning, especially in model checking: to check if a system S satisfies a DC property D, we can check a sampling system S_h of S whether it satisfies a discrete DC property D_h. D_h is found such that $S_h \models D_h$ implies $S \models D$. This technique has been used in [48].

Periodic Task System Now we return to the scheduler mentioned in the introduction of the paper. Recall that a periodic task system T consists of n processes $\{1,\ldots,n\}$. Each process i raises its request periodically with period T_i, and for each period it requests a constant amount of processor time C_i. A specification of system T in DC has been given in many works, see e.g [70], which assume that all the processes raise their request at time 0. Now we can give a complete specification of the system without this assumption using the same technique that was introduced for temporal variable *int* in the previous section. To specify periodic behaviour of process i, we also use temporal variable $dLine\_i$ as in [70] whose behavior is similar to temporal variable *int*, and specified by:

$$dLine\_i \Rightarrow ((dLine\_i \wedge \ell = 0)^\frown(dLine\_i \wedge \ell = T_i)^*) \wedge ((dLine\_i \wedge \ell = T_i)^*{}^\frown(dLine\_i \wedge \ell = 0)) \tag{11}$$

$$dLine\_i{}^\frown(\ell = T_i) \Rightarrow dLine_i \tag{12}$$

$$\ell \geq 2T_i \Rightarrow \ell < T_i{}^\frown((dLine\_i \wedge \ell = T_i)^* \wedge \neg(true{}^\frown(dLine\_i \wedge \ell = T_i)^\frown\neg(dLine\_i \wedge \ell = T_i)^*) \wedge \neg(\neg(dLine\_i \wedge \ell = T_i)^*{}^\frown(dLine\_i \wedge \ell = T_i)^\frown true)^\frown \ell < T_i \tag{13}$$

Let Run_i be a state variable saying that process i is running on the processor, i.e. $Run_i(t) = 1$ if process i is running on the processor, and $Run_i(t) = 0$ otherwise. Let $Stand_i$ be a state variable saying that the current request of process i has not been fulfilled. The behaviour of process i is fully specified by the following formula $\mathcal{B}_i$:

$$dLine\_i \wedge \ell = T_i \Rightarrow (((\int Run_i < C_i \Leftrightarrow \lceil\!\lceil Stand_i \rceil\!\rceil)^\frown true) \wedge (\int Run_i = C_i{}^\frown \ell > 0 \Rightarrow \int Run_i = C_i{}^\frown \lceil\!\lceil \neg Stand_i \rceil\!\rceil))$$

The requirement REQ of system T is simply specified by:

$$\wedge_{i\leq n} dLine\_i \wedge \ell = T_i \Rightarrow \int Run_i = C_i$$

Denote by $PERIOD$ the conjunction of formulas 11, 12 and 13. $PERIOD$ forms a complete specification of temporal propositional variables $dLine_i$, $i \leq n$, and are useful in proving the correctness of a scheduler for system T.

A priority-based scheduler $\mathcal{S}$ for system T with single processor is characterised by state variables $HiPri_{ij}$ $(i, j \leq n, i \neq j)$ which specify the dynamic priority among the processes defined by $\mathcal{S}$, and the following state formulas characterising its behaviour:

$$\begin{array}{l}
\wedge_{i\neq j}((Run_i \wedge Stand_j) => HiPri_{ij}) \\
\wedge_{i\leq n}(Run_i => Stand_i) \\
\wedge_{i\neq j}(HiPri_{ij} \Rightarrow \neg HiPri_{ji}) \\
\wedge_{i\neq j}\neg(Run_i \wedge Run_j) \\
\vee_{i\leq n} Stand_i \Rightarrow \vee_{i\leq n} Run_i
\end{array}$$

The first formula says that a standby process j is not running because there is a process i with higher priority is running. The second formulas says that only standby process can run, the third formula characterises that the priority relation is totally defined, and the last formula specifies that only if there is a standby process then at least one process should be running. Let SCH denote the conjunction of these five formulas.

Deadline driven scheduler is a priority-based scheduler that considers process i to have a higher priority than process j (i.e. the value of $HiPri_{ij}$ at the current time point is 1) iff the deadline for process i is nearer than the deadline for process j. The deadline driven scheduler can be modelled with the additional formula specifying the behaviour of state variables $HiPri_{ij}$ $(i, j \leq n)$:

$$\wedge_{i\neq j} \lceil HiPri_{ij} \rceil^{\frown} \ell = T_i \Rightarrow (\neg \Diamond dLine_j)^{\frown} dLine_i^{\frown} true$$

Denote this formula by DDS. The interesting thing here is that variables $HiPri_{ij}$ can be defined in DC, without any quantification on rigid variables, via temporal propositional variables $dLine_i$ $(i \leq n)$ which are completely specified by formulas 11, 12 and 13. Note that with defining $HiPri_{ij}$ in this way, we don't have to assume that all the processes raise their request at time 0. Hence, reasoning about the correctness of the scheduler for the general case can be done with the proof system of DC. For example, the correctness of deadline-driven scheduler (included in Liu and Layland's Theorem for the feasibility of the deadline-driven scheduler) is formalised as:

$$\{PERIOD, \wedge_{i\leq n}\mathcal{B}_i, SCH, DDS, \textstyle\sum_{i\leq n}(C_i/T_i) \leq 1\} \models REQ$$

This can be proved by using the proof system of DC.

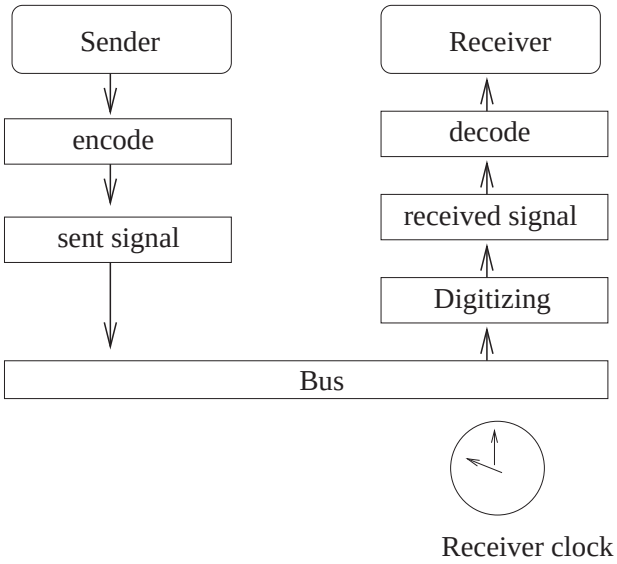


Fig. 6. Communication Protocol Model

8.3 Modelling Communication Protocols with Digitizing in DC

In this section, we show that with discrete time structure formalised, we can model communication protocols using Duration Calculus (DC) in a very convenient way without any extension for digitising. This model has been presented in our earlier work [32, 33]. Consider a model for communication at the physical layer (see Fig. 6). A sender and a receiver are connected via a bus. Their clocks are running at different rates. We refer to the clock of the receiver as the time reference. The receiver receives signals by digitising. Since the signals sent by the sender and the signals received by the receiver are functions from the set $\mathbb{R}^+$ to $\{0, 1\}$ (1 represents that the signal is *high*, and 0 represents that the signal is *low*), we can model them as state variables in DC.

The communication protocols are modelled in DC as follows. The signal sent by the sender is modelled by a state X. The signal received by the receiver by sampling the signal on the bus is modelled by a state Y in the sampling time model with the sampling time step 1. So, $step \Leftrightarrow int \wedge \ell = 1$. However, it is not the case that $samp(X, Y)$ due to the fact that it takes a significant amount of time to change the signal on the bus from high to low or vice-versa, and hence, the signal on the bus cannot be represented by a Boolean function. Without loss of generality, assume that the delay between the sender and the receiver is 0. Assume also that when the signal on the bus is neither high nor low, the receiver will choose an arbitrary value from $\{0, 1\}$ for the value of Y. The phenomenon is depicted in Fig. 7. Assume that it takes r (r is a natural number) receiver-clock cycles for the sender to change the signal on the bus from high to low

or vice-versa. Then if the sender changes the signal from low to high or from high to low, the receiver's signal will be unreliable for r cycles starting from the last tick of the receiver clock and during this period it can be any value chosen nondeterministically from 0 and 1. Otherwise, the signal received by the receiver is the same as the signal sent by the sender (see Figure 7). This relationship between X and Y is formalised as

$$(\lceil X \rceil \wedge (\ell \geq r+1)) \Rightarrow (\ell \leq r)^\frown(\lceil Y \rceil \wedge int)^\frown(\ell < 1)\,,$$
$$(\lceil \neg X \rceil \wedge (\ell \geq r+1)) \Rightarrow (\ell \leq r)^\frown(\lceil \neg Y \rceil \wedge int)^\frown(\ell < 1)\,.$$

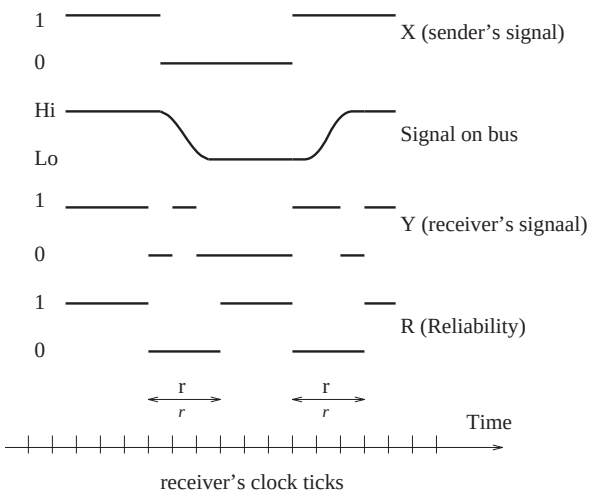


Fig. 7. Signal patterns

Since the behaviour of a state can be specified by a DC formula, a communication protocol can be modelled as consisting of a coding function f, which maps a sequence of bits to a DC formula expressing the behaviour of X, and a decoding function g, which maps a DC formula expressing the behaviour of Y to a sequence of bits. The protocol is correct iff for any sequence w of bits, if the sender puts the signal represented by $f(w)$ on the bus then by digitising the receiver must receive and receives only the signals represented by a DC formula D for which $g(D) = w$.

8.4 Biphase Mark Protocols

In the Biphase Mark Protocols (BMP) the sender encodes a bit as a cell consisting of a mark subcell of length b and a code subcell of length a. The sender keeps the signal stable in each subcell (hence either $\lceil X \rceil$ or $\lceil \neg X \rceil$ holds for the interval representing a subcell). For a cell, if the signal in the mark subcell is

the same as the signal in the code subcell, the information carried by the cell is 0; otherwise, the information carried by the cell is 1. There is a phase reverse between two consecutive cells. This means that, for a cell, the signal of the mark subcell of the following cell is held as the negation of the signal of the code subcell of the cell. The receiver, on detecting a state change (of Y), knows that it is the beginning of a cell, and skips d cycles (called the *sampling distance*) and samples the signal. If the sampled signal is the same as the signal at the beginning of the cell, it decodes the cell as 0; otherwise it decodes the cell as 1.

At the beginning of the transmission, the signal is low for a cycles (this means, $[\![\neg X]\!]$ holds for the interval of length a starting from the beginning). When the sender finishes sending, it keeps the signal stable for cc time units which is longer than the code subcell. We use HLS, LHS to denote the formulas representing intervals consisting of the code subcell of a cell and the mark subcell of the next one for the sender, and use $HLR^\frown(\ell = d)$, $LHR^\frown(\ell = d)$ to denote the formulas representing the intervals between the two consecutive sampling points (from the time the receiver samples the signal of a code subcell to the next one. Formally,

$$
\begin{aligned}
HLS &\mathrel{\widehat{=}} ([\![X]\!] \wedge \ell = a)^\frown([\![\neg X]\!] \wedge \ell = b)\,,\\
LHS &\mathrel{\widehat{=}} ([\![\neg X]\!] \wedge \ell = a)^\frown([\![X]\!] \wedge \ell = b)\,,\\
HLR &\mathrel{\widehat{=}} ([\![Y]\!] \wedge int \wedge 1 \leq \ell \leq \rho)^\frown([\![\neg Y]\!] \wedge \ell = 1)\,,\\
LHR &\mathrel{\widehat{=}} ([\![\neg Y]\!] \wedge int \wedge 1 \leq \ell \leq \rho)^\frown([\![Y]\!] \wedge \ell = 1)\,.
\end{aligned}
$$

Now, we are ready to formalise the BMP in DC. What we have to do is write down the encoding function f and the decoding function g. From the informal description of the protocol, we can define f inductively as follows.

1. $f(\epsilon) \mathrel{\widehat{=}} ([\![\neg X]\!] \wedge \ell = c)$
2. If $f(w) = D^\frown([\![X]\!] \wedge \ell = c)$, then

$$
\begin{aligned}
f(w0) &\mathrel{\widehat{=}} D^\frown HLS^\frown([\![\neg X]\!] \wedge \ell = c)\\
f(w1) &\mathrel{\widehat{=}} D^\frown HLS^\frown([\![X]\!] \wedge \ell = c)
\end{aligned}
$$

3. If $f(w) = D^\frown([\![\neg X]\!] \wedge \ell = c)$, then

$$
\begin{aligned}
f(w0) &\mathrel{\widehat{=}} D^\frown LHS^\frown([\![X]\!] \wedge \ell = c)\\
f(w1) &\mathrel{\widehat{=}} D^\frown LHS^\frown([\![\neg X]\!] \wedge \ell = c)
\end{aligned}
$$

For example, $f(1) = LHS^\frown([\![\neg X]\!] \wedge \ell = c)$, $f(10) = LHS^\frown LHS^\frown([\![X]\!] \wedge \ell = c)$, and $f(101) = LHS^\frown LHS^\frown HLS^\frown([\![X]\!] \wedge \ell = c)$.

Because the decoding function g is a partial function, we have to describe its domain first, i.e. what kind of DC formulas on the state Y are detected (received) by the receiver. According to the behaviour of the receiver, first it skips r cycles. Then it begins to scan for an edge (HLR or LHR). When an edge is detected, it skips d cycles and repeats this procedure until it detects that the transmission has completed (Y is stable for more than ρ cycles). Thus, a DC formula D is received by the receiver iff D is of the form $A_0{}^\frown A_1{}^\frown \ldots {}^\frown A_n$, $n \geq 1$, where

- $A_0 = (1 \geq \ell \wedge \ell > 0)^\frown(int \wedge (\ell = r - 1)))$

– and either $A_n = (int \wedge \lceil Y \rceil \wedge (\ell > \rho))^\frown(\ell < 1))$,
or $A_n = (int \wedge \lceil \neg Y \rceil \wedge (\ell > \rho))^\frown(\ell < 1))$
– and for $j = 1, \ldots, n-1$ either $A_j = LHR^\frown(\ell = d)$ or $A_j = HLR^\frown(\ell = d)$
– and if $n = 1$ then $A_n = (int \wedge \lceil \neg Y \rceil \wedge (\ell > \rho))^\frown(\ell < 1))$ and if $n > 1$ then $A_1 = LHR^\frown(\ell = d)$ (since at the beginning the signal is low).

Now, the decoding function g can be written as follows. Let D be a formula received by the receiver.

– If $D = (\ell \leq 1 \wedge \ell > 0)^\frown(int \wedge \ell = r-1)^\frown(\lceil \neg Y \rceil \wedge \ell > \rho \wedge int)^\frown \ell < 1$ then $g(D) = \epsilon$.
– Let $g(D)$ be defined.
  - If $D = D'^\frown(\lceil Y \rceil \wedge int \wedge \ell \geq \rho)^\frown \ell < 1$ then
    $g(D'^\frown HLR^\frown(\ell = d)^\frown(\lceil Y \rceil \wedge int \wedge \ell \geq \rho)^\frown \ell < 1) = g(D)1$, and
    $g(D'^\frown HLR^\frown(\ell = d)^\frown(\lceil \neg Y \rceil \wedge int \wedge \ell \geq \rho)^\frown \ell < 1) = g(D)0$.
  - If $D = D'^\frown(\lceil \neg Y \rceil \wedge int \wedge \ell \geq \rho)^\frown \ell < 1$, then
    $g(D'^\frown LHR^\frown(\ell = d)^\frown(\lceil Y \rceil \wedge int \wedge \ell \geq \rho)^\frown \ell < 1) = g(D)0$, and
    $g(D'^\frown LHR^\frown(\ell = d)^\frown(\lceil \neg Y \rceil \wedge int \wedge \ell \geq \rho)^\frown \ell < 1) = g(D)1$.

For example, let D be $(\ell \leq 1 \wedge \ell > 0)^\frown(int \wedge \ell = r-1)^\frown LHR^\frown(\ell = d)^\frown LHR^\frown(\ell = d)^\frown HLR^\frown(\ell = d)^\frown(\lceil Y \rceil \wedge \ell > \rho \wedge int)^\frown(\ell < 1))$. Then,

$$\begin{aligned}
g(D) &= g((\ell \leq 1 \wedge \ell > 0)^\frown(int \wedge \ell = r-1)^\frown LHR^\frown(\ell = d) \\
&\quad \frown LHR^\frown(\ell = d)^\frown(\lceil Y \rceil \wedge \ell > \rho \wedge int)^\frown(\ell < 1))\,1 \\
&= g((\ell \leq 1 \wedge \ell > 0)^\frown(int \wedge \ell = r-1)^\frown LHR^\frown(\ell = d)^\frown \\
&\quad (\lceil \neg Y \rceil \wedge \ell > \rho \wedge int)^\frown(\ell < 1))\,01 \\
&= g((\ell \leq 1 \wedge \ell > 0)^\frown(int \wedge \ell = r-1) \\
&\quad \frown(\lceil \neg Y \rceil \wedge \ell > \rho \wedge int)^\frown(\ell < 1))\,101 \\
&= \epsilon 101\,.
\end{aligned}$$

8.5 Verification of BMP

As said earlier, we have to verify that for any sequence of bits w, if the sender puts on the bus the signal represented by DC formula $f(w)$, then the receiver must receive and receives only the signals represented by a DC formula D for which $g(D) = w$. We can only prove this requirement with some condition on the values of the parameter r, a, b, c, ρ and d. The requirement is formalised as:

For all sequence of bits w,

– there exists a DC formula D received by the receiver such that $f(w) \Rightarrow D$, and
– for all D receivable by the receiver, if $f(w) \Rightarrow D$ then $g(D) = w$.

Since in BMP g is a deterministic function, for any sequence of bits w there is no more than one receivable formula D for which $f(w) \Rightarrow D$. Thus we can have a stronger requirement which is formalised as:

For all sequences of bits w there exists uniquely a receivable formula D such that $f(w) \Rightarrow D$ and $g(D) = w$.

Our verification is done by proving the following two theorems.

Theorem 22. *[31] For any receivable formulas D and D', if D is different from D' syntactically then $\models ((D \wedge D') \Rightarrow ff)$.*

This theorem says that each time at most one receivable formula D is received by the receiver.

Theorem 23. *[31] Assume that $r \geq 1$, $b \geq r+1$, $a \geq r+1$, $c \geq \rho+a$, $d \geq b+r$, $d \leq a+b-3-r$, and $\rho \geq a+1$. Then for any sequence of bits w there exists a receivable formula D for which $f(w) \Rightarrow D$ and $g(D) = w$.*

In [33] we proved these two theorems, with PVS proof checker, with the encoding of the proof system for Duration Calculus.

We have seen in this section that by using temporal propositional letters we can specify many classes of time models that are suitable for our applications. The properties of the introduced temporal propositional letters are then specified by a class of Duration Calculus formulas. Using this class of formulas and the proof system of the original Duration Calculus we can reason about the behaviour of our real-time systems in different time domains.

9 Conclusion

We have presented in this chapter a theory of Duration Calculus with applications. This theory contains the main components of the calculus such as its syntax, semantics and proof system. We also present some extensions of DC which are convenient for specification and give more expressive power to DC. We give a brief summary of the decidability and undecidability results and present some techniques for doing model-check for a sub-class of the calculus that have been published in the literatures, and for specifying real-time properties of computing systems via some well-known examples. The materials presented in this chapter is fundamental for researching and practicing in Duration Calculus. The research in this area is on the way, and we believe that many newly developed results in Duration Calculus are not covered in this chapter.

References

1. E. Asarin, P. Caspi and O. Maler. A Kleene Theorem for Timed Automata. International Symposium on Logics in Computer Science LICS'97. IEEE computer Society Press, 1997, pp. 160–171.
2. Allen JF (1984) Towards a general theory of action and time. Artificial Intelligence 23:123 – 154
3. Alur R, Dill DL (1994). A theory of timed automata. Theoretical Comput. Sci., 126(2):183–235
4. Bengtsson J, Larsen KG, Larsson F, Pettersson P, and Yi W (1997) Uppaal – a tool suite for automatic verification of real-time systems. In Alur R, Henzinger T, Sonntag E (eds) Hybrid Systems III – Verification and Control, LNCS 1066, Springer, Berlin Heidelberg New York: 232–243

5. Bjørner D (1992) Trusted computing systems: the ProCoS experience. In Proceedings of the 14th international conference on Software engineering, Melbourne, Australia, ACM Press: 15 – 34
6. Bjørner D, Hensson M (eds) (2007) *Logics of Specification Languages*, EATCS Monographs in Theoretical Computer Science, Springer (to appear)
7. Blackburn P, Rijke M de, Venema Y (2001) Modal Logic. Cambridge
8. Bolander B, Hansen JU, Hansen MR (2006) Decidability of hybrid duration calculus. ENTCS 174(6), pp. 113-133, Elsevier 2007
9. Braberman VA, Hung DV (1998) On checking timed automata for linear duration invariants. In Proceedings of the 19th IEEE Real-Time Systems Symposium, IEEE Computer Society Press: 264 – 273
10. Buchi JR (1960) Weak second-order arithmetic and finite automata. Z. Math. Logik Grundl. Math 6:66–92
11. Chan P, Hung DV (1995) Duration calculus specification of scheduling for tasks with shared resources. In Kanchanasut K, Levy JJ (eds) Asian Computing Science Conference 1995, LNCS 1023, Springer, Berlin Heidelberg New York: 365–380
12. Dutertre B (1995) Complete proof systems for first order interval temporal logic. In Tenth Annual IEEE Symp. on Logic in Computer Science, IEEE Press:36–43
13. Dutertre B (1995) On first order interval temporal logic. Technical report, Report no. CSD-TR-94-3, Department of Computer Science, Royal Holloway, University of London, Eghan, Surrey TW20 0EX, England
14. Elgot CC (1961) Decision problems of finite automata design and related arithmetics. Transactions of the American Mathematical Society 98:21–52
15. Fränzle M, Hansen MR (2005) A Robust Interpretation of Duration Calculus. In Hung DV, Wirsing M (eds) Theoretical Aspects of Computing – ICTAC 2005, LNCS 3722, Springer, Berlin Heidelberg New York: 257–271
16. Fränzle M, Hansen MR (2007) Deciding an Interval Logic with Accumulated Durations. In Grumberg O, Huth M DV, (eds) Tools and algorithms for the construction and analysis of systems – TACAS 2007, LNCS 4424, Springer, Berlin Heidelberg New York: 201–215
17. Fränzle (1996) Synthesizing controllers from duration calculus. In Jonsson B, Parrow J (eds) Formal Techniques in Real-Time and Fault-Tolerant Systems, LNCS 1135, Springer, Berlin Heidelberg New York:168–187
18. Fränzle M (2002) Take it np-easy: Bounded model construction for duration calculus. In Damm W, Olderog ER (eds) Formal Techniques in Real-Time and Fault-Tolerant Systems, LNCS 2469, Springer, Berlin Heidelberg New York:245–264
19. Guelev DP and Dang Van Hung. On the completeness and decidability of duration calculus with iteration. *Theor. Comput. Sci.*, 337(1-3):278–304, 2005.
20. Guelev DP. A Complete Proof System for First Order Interval Temporal Logic with Projection. Technical Report 202, UNU-IIST, P.O.Box 3058, Macau, June 2000. A revised version of this report was published in the Journal of Logic and Computation, Volume 14, Issue 2, April 2004, pp. 215-249 by Oxford University Press.
21. Guelev DP (1998) A calculus of durations on abstract domains: Completeness and extensions. Technical report, UNU/IIST 139
22. Guelev DP, Hung DV (1999) On the completeness and decidability of duration calculus with iteration. In Thiagarajan S, Yap R (eds) Advances in Computing Science, LNCS 1742, Springer, Berlin Heidelberg New York:139–150

23. Halpern J, Moskowski B, Manna Z (1983) A hardware semantics based on temporal intervals. In ICALP'83, LNCS 154, Springer, Berlin Heidelberg New York:278–291
24. Halpern J, Shoham Y (1991) A propositional modal logic of time intervals. Journal of the ACM 33(4):935–962
25. Hansen MR (2007). Duration Calculus. Chapter in [6]: pp. 293-441
26. Hansen MR (1994) Model-checking discrete duration calculus. Formal Aspects of Computing 6A:826–845
27. Hansen MR, Zhou C (1992) Semantics and completeness of duration calculus. In Bakker JW de, Huizing C, Roever WP de, Rozenberg G (eds) Real-Time: Theory in Practice, LNCS 600, Springer, Berlin Heidelberg New York:209–225
28. Hansen MR, Zhou C (1997) Duration calculus: Logical foundations. Formal Aspects of Computing 9:283–330
29. Harrison J (1998) Theorem Proving with the Real Numbers. Springer, Berlin Heidelberg New York
30. Hoenicke (2006) Combination of Processes, Data and Time. Dissertation, Carl von Ossietzky Universität, Oldenburg, Germany
31. Dang Van Hung. Specifying Various Time Models with Temporal Propositional Variables in Duration Calculus. Research Report 377, UNU-IIST, P.O.Box 3058, Macau, June 2007.
32. Hung DV and Il K-K. Verification via Digitized Model of Real-Time Systems. Research Report 54, UNU-IIST, P.O.Box 3058, Macau, February 1996. Published in the Proceedings of *Asia-Pacific Software Engineering Conference 1996* (APSEC'96), IEEE Computer Society Press, 1996, pp. 4–15.
33. Hung DV (1998) Modelling and verification of biphase mark protocols in duration calculus using pvs/dc$^-$. In Application of Concurrency to System Design (CSD'98), IEEE Computer Society Press:88–98
34. Hung DV and Thai PH. Checking a Regular Class of Duration Calculus Models for Linear Duration Invariants. Technical Report 118, UNU/IIST, P.O.Box 3058, Macau, July 1997. Presented at and published in the Proceedings of the International Symposium on Software Engineering for Parallel and Distributed Systems (PDSE'98), 20 - 21 April 1998, Kyoto, Japan, Bernd Kramer, Naoshi Uchihira, Peter Croll and Stefano Russo (Eds), IEEE Computer Society Press, 1998, pp. 61 - 71.
35. Hung DV, Giang PH (1996) A sampling semantics of duration calculus. In Jonsson B, Parrow J (eds) Formal Techniques for Real-Time and Fault Tolerant Systems, LNCS 1135. Springer, Berlin Heidelberg New York: 188–207
36. Hung DV, Guelev DP (1999) Completeness and decidability of a fragment of duration calculus with iteration. In Thiagarajan PS, Yap R (eds) Advances in Computing Science, LNCS 1742. Springer, Berlin Heidelberg New York:139–150
37. Hung DV, Ji W (1996) On design of hybrid control systems using i/o automata models. In Chandru V, Vinay V (eds) Foundations of Software Technology and Theoretical Computer Science, LNCS 1180. Springer, Berlin Heidelberg New York:156–167
38. Zhao J, Hung DV (1998) On checking real-time parallel systems for linear duration properties. In Ravn AP, Rischel H (eds) Formal Techniques in Real-Time and Fault-Tolerant Systems, LNCS 1486. Springer, Berlin Heidelberg New York:241–250
39. Kesten Y, Pnueli A, Sifakis J, Yovine S (1993) Integration graphs: A class of decidable hybrid systems. In Grossman RL, Nerode A, Ravn AP, Rischel H (eds) Hybrid Systems, LNCS 736. Springer, Berlin Heidelberg New York:179–208

40. Klarlund N, Moller A (2001) MONA Version 1.4: User Manual. BRICS, Department of Computer Science, University of Aarhus, Denmark
41. Larsen KG, Rasmussen JI (2005) Optimal conditional reachability for multipriced timed automata. In Sassone V, editor, Foundations of Software Science and Computation Structures (FOSSACS '05), LNCS 3441, Springer, Berlin Heidelberg New York:230–244
42. Li X (1993) A Mean Value Calculus. PhD Thesis, Software Institute, Academia Sinica
43. Li X, Hung DV (1996) Checking linear duration invariants by linear programming. In Jaffar J, Roland H, Yap C (eds) Concurrency and Parallelism, Programming, Networking, and Security, LNCS 1179. Springer Berlin Heidelberg New York:321–332
44. Li X, Hung DV, Zheng T (1997) Checking hybrid automata for linear duration invariants. In Advances in Computing Science, LNCS 1997. Springer Berlin Heidelberg New York:166–180
45. Mao X, Xu Q, Hung DV, Wang J (1996) Towards a proof assistant for interval logics. Technical report, UNU/IIST Report No. 77, UNU/IIST, International Institute for Software Technology, Macau
46. Moszkowski (1985) A temporal logic for multilevel reasoning about hardware. IEEE Computer 18(2):10–19
47. Olderog ER, Dierks H (1998) Decomposing real-time specifications. In Langmaack H, Roever WP de, Pnueli A (eds) Compositionality: The Significant Difference, LNCS 1536, Springer, Berlin Heidelberg New York
48. Pandya PK, Krishna SN, and Loya K. On Sampling Abstraction of Continuous Time Logic with Duration Calculus. Technical Report TIFR-PKP-GM-2006/1, Tata Institute of Fundamental Research, India, 2006.
49. Pandya PK (1996) Some extensions to propositional mean value calculus: Expressiveness and decidability. In Kleine Buening H (ed) CSL'95, LNCS 1092. Springer, Berlin Heidelberg New York:434–451
50. Pandya PK (2000) Specifying and deciding quantified discrete-time duration calculus formulae using DCVALID. Tata Institute of Fundamental Research, India. Technical report, TCS00-PKP-1
51. Pandya PK (1999) Dcvalid 1.3: The user manual. Technical report, Computer Science Group, TIFR, Bombay, Technical Report TCS-99/1
52. Rabinovich A (1998) Non-elementary lower bound for propositional duration calculus. Information Processing Letters:7-11
53. Rabinovich A (1998) On the decidability of continuous time specification formalism. Journal of logic and computation 8(5):669-678.
54. Rasmussen TM (2001) Automated proof support for interval logics. In LPAR 2001, LNAI 2250. Springer Berlin Heidelberg New York:317–326
55. Rasmussen TM (2001) Labelled natural deduction for interval logics. In Computer Science Logic, CSL'01, LNCS 2142. Springer Berlin Heidelberg New York:308–323
56. Rasmussen TM (2002) Interval Logic: Proof Theory and Theorem Proving. PhD Thesis, Informatics and Mathematical Modelling, Technical University of Denmark
57. Ravn AP (1995) Design of Embedded Real-Time Computing Systems. Doctoral Dissertation, Department of Computer Science, Technical University of Denmark
58. Ravn AP, Rischel H, Hansen KM (1993) Specifying and verifying requirements of real-time systems. IEEE Trans. Softw. Eng. 19(1):41–55

59. Satpathy M, Hung DV, Pandya PK (1998) Some results on the decidability of duration calculus under synchronous interpretation. In Ravn AP,Rischel H (eds) Formal Techniques in Real-Time and Fault-Tolerant Systems, LNCS 1486. Springer, Berlin Heidelberg New York:186–197
60. Skakkebæk JU (1994) A Verification Assistant for a Real-Time Logic. PhD Thesis, Department of Computer Science, Technical University of Denmark
61. Skakkebæk JU, Sestoft P (1994) Checking validity of duration calculus formulas. Technical report, ProCoS II, ESPRIT BRA 7071, report no. ID/DTH JUS 3/1, Department of Computer Science, Technical University of Denmark
62. Skakkebæk JU, Shankar N (1994) Towards a duration calculus proof assistant in pvs. In Langmack H, Roever WP de, Vytopil J (eds) Techniques in Real-Time and Fault-Tolerant Systems, LNCS 863. Springer, Berlin Heidelberg New York:660–679
63. Sørensen EV, Ravn AP, Rischel H (1990) Control program for a gas burner: Part 1: Informal requirements, procos case study 1. Technical report, ProCoS Rep. ID/DTH EVS2
64. Tarski A (1948) A decision method for elementary algebra and geometry. RAND Corporation, Santa Monica, California
65. Thai PH and Hung DV. Verifying Linear Duration Constraints of Timed Automata. Presented at and published in the proceedings of the First International Colloquium on Theoretical Aspects of Computing ICTAC'04, Gui Yang, China, 22-24 September, 2004, LNCS 3407, pp. 295-309, Springer-Verlag, 2005.
66. F. Wang, A. K. Mok, and E. A. Emerson. Distributed Real-Time System Specification and Verification in APTL. *ACM Transactions on Software Engineering and Methodology*, 2(4):346–378, October 1993.
67. Zheng Y, Zhou C (1994) A formal proof of the deadline driven scheduler. In Langmack H, Roever WP de, Vytopil J (eds) Techniques in Real-Time and Fault-Tolerant Systems, LNCS 863. Springer, Berlin Heidelberg New York:756–775
68. Zhou C, Hansen MR (1998) An adequate first order logic of intervals. In Langmaack H, Roever WP de, Pnueli A (eds) Compositionality: The Significant Difference, LNCS 1536. Springer Berlin Heidelberg New York:584–608
69. Zhou C, Hansen MR, Ravn AP, Rischel (1991) Duration specifications for shared processors. In Vytopil J (ed) Symposium on Formal Techniques in Real-Time and Fault Tolerant Systems, LNCS 571. Springer Berlin Heidelberg New York:21–32
70. Zhou C, Hansen MR(2004) Duration Calculus: A formal approach to real-time systems. Springer, Berlin Heidelberg New York
71. Zhou C, Hansen MR, Sestoft P (1993) Decidability and undecidability results for duration calculus. In Enjalbert P, Finkel A, Wagner KW (eds) STACS'93, LNCS 665. Springer Berlin Heidelberg New York:58–68
72. Zhou C, Hoare CAR, Ravn AP (1991) A calculus of durations. Information Processing Letters 40(5):269–276
73. Zhou C, Zhang J, Yang L, Li X (1994) Linear duration invariants. In Langmack H, Roever WP de, Vytopil J (eds) Techniques in Real-Time and Fault-Tolerant Systems, LNCS 863. Springer, Berlin Heidelberg New York:86–109