A Higher-Order Duration Calculus and Its Completeness [1]

Zhan Naijun
Lab. of Computer Science and Technology, Institute of Software,
the Chinese Academy of Sciences, Beijing, 100080, P.R. China
Email: znj@ox.ios.ac.cn

Abstract This paper studies how to describe the real-time behaviour of programs using Duration Calculus. Since program variables are interpreted as functions over time in real-time programming, and it is inevitable to introduce quantifications over program variables in order to describe local variable declaration and declare local channel and so on, therefore, to establish a higher-order duration calculus (HDC) is necessary. We firstly establish HDC, then show some real-time properties of programs in terms of HDC, and lastly, prove that HDC is complete on abstract domains under the assumption that all program variables vary finitely in the paper.

Keywords: duration calculus higher-order logics real-time programs super-dense computation completeness

[1] proposed a first order interval logic called Duration Calculus (DC), which is an extension of interval temporal logic [2] (ITL), and can specify real-time requirements of computing systems. This logic has been used to specify real-time requirements of computing systems, including software embedded systems; On the other hand, in order to establish a programming methodology based on DC, DC has been applied to define real-time semantics of programming languages. In all these attempts, local variables and internal channels have not been taken into account, since quantifications over temporal variables have not been investigated. In general, higher-order logic is much more complicated than first order one. Detail introduction to its applications can be seen [3]. This paper investigates this issue and establishes a higher-order duration calculus by introducing quantifications over programs, then shows some real-time properties of programs using it, and finally, proves it is complete on abstract domains if all program variables vary finitely is assumed. A by-product of the higher-order logic is that [4] points out that the super-dense chop [6] which is used to deal with super-dense computation [5] can be derived from the higher-order quantifications.

We will omit the proofs for some theorems in order to save space, their proofs can be seen in [3]. Some results of the paper also appear in [7, 8].

1 Introduction

1.1 Higher-order Duration Calculus and Its Completeness

In order to specify the behaviour of real-time programs, program variables V, V_i are introduced into HDC as temporal variables. Program variables, global variables and constants are state terms, and real arithmetic functions applying to state terms are also state terms. Predicates of program variables are taken as states: $(\mathsf{R} \rightarrow \{0,1\})$, if S_1 and S_2 are states, then $\neg S_1$, $S_1 \vee S_2$ and $S_1 \wedge S_2$ are also states. Hence, states in HDC have internal structures, e.g. $(V = x)$ and $(V > 1)$ can be taken as states.

Value passing involves past and future time, to receive an initial value from the previous statement and to pass final value to the next statement. The chop modality is a contracting one, and cannot express state properties outside current interval. Therefore, two special functions "$\leftarrow$" and

[1]The work is partially supported by UNU/IIST and the National Natural Science Foundation of China under grant No. 69873003.

"→" [11], which have a domain of state terms and a co-domain of functions from intervals to duration domain and are used to calculate the values of state terms in left and right neighbourhood of the current interval respectively, are introduced into HDC. E.g. $(\overleftarrow{V}= 2)$ means that in a left neighbourhood of the current interval the value of V is 2. So it has the following property

$$(\ell > 0)^\frown(\overleftarrow{V}= 2) \ \Rightarrow\ (true^\frown\lceil V = 2\rceil)^\frown(\overleftarrow{V}= 2)$$

where $\lceil S\rceil \,\widehat{=}\, \int S = \ell \wedge \ell > 0$ means that S presents almost everywhere in a non-point interval. We will use $\lceil\ \rceil$ to denote $\ell = 0$. Symmetrically, $(\overrightarrow{V}= 2)$ means that in a right neighbourhood of the current interval the value of V is 2. Hence

$$(\overrightarrow{V}= 2)^\frown(\ell > 0) \ \Rightarrow\ (\overrightarrow{V}= 2)^\frown(\lceil V = 2\rceil^\frown true)$$

The semantics of an assignment $(V := V + 2)$, if we only take care of program variable V, can be described as receiving the value of V from the previous statement, adding 2 to it, and then passing the updated one to the following statement. That is, the semantic definition $[\![V := V + 2]\!]$ can be formulated as

$$\exists x.\ (\overleftarrow{V}= x) \ \wedge\ \lceil V = x + 2\rceil \ \wedge\ (\overrightarrow{V}= x + 2)$$

where we have assumed that the assignment $(V := V + 2)$ consumes certain time, so that the state $(V = x + 2)$ will be stable.

In both interval temporal logic [9] and duration calculi [1, 10, 11, 12], symbols are divided into flexible and rigid symbols (adopting the terminology of [13, 14]). Rigid symbols are intended to represent fixed, global entities. Their interpretation will be the same in all intervals. Conversely, entities which may vary in different intervals are represented by flexible symbols. Such a distinction between two classes of symbols is common in the context of first order temporal logics [13, 14].

The completeness of interval temporal logics and duration calculi not only depends on the choice of time domain, but also relies on which kind of variables are quantified. In practice, we like to look the reals as time domain. If so, we cannot get the completeness of these systems, for if they were, they would be adequate for arithmetic, which is impossible by Gödel's Theorem. Therefore, if we want to choose the reals as time domain, we can only get the relative completeness of these systems. E.g. the relative completeness of DC has been proved in [15]. If we only quantify over global variables, duration calculi are complete on abstract domains [16]. But if we introduce quantifications over program variables into DC, since we interpret program variables as functions from time domain to duration domain, no (consistent) system is complete for this semantics because whenever we interpret the domain of a quantifier as the set of all functions from time domain to duration domain, the language will have the expressive power of second-order arithmetic, but it is impossible by the result that Gödel's Theorem applies. So some restrictions on program variables are necessary in order to work out a complete proof system. Therefore, we assume that all program variables vary finitely. This is consistent with the assumption that all state variables have finite variability in the original DC. The paper will prove that HDC is complete on abstract domains under the above assumption by reducing HDC to a complete first-order two-sorted interval temporal logic.

1.2 Real-time Semantics of Programs

The paper will also demonstrate how to give a real-time semantics of programs in terms of HDC, particularly, how to define super-dense chop in HDC.

It is a comfortable abstraction to assume that a sequence of program statements is executed one by one, but consumes no time by comparison with an environment of a grand time granularity. A computation of a sequence of operations which is assumed timeless is called a super-dense computation [5].

Under the assumption of the super-dense computation, $[\![V := V + 2]\!]$ will become

$$\exists x.\ (\overleftarrow{V} = x)\ \wedge\ \lceil\ \rceil\ \wedge\ (\overrightarrow{V} = x + 2)$$

since the state of $(V = x + 2)$ vanishes immediately.

By the meaning of the chop modality, it seems natural to denote the sequential composition (;) by the chop. Hence, the semantics of the two consecutive assignments

$$V =: V + 2; V =: V + 1$$

would be denoted by

$$[\![V =: V + 2]\!]^\frown[\![V := V + 1]\!]$$

Unfortunately, the chop modality at a point interval degenerates into the conjunction, and the above formula is changed to

$$[\![V =: V + 2]\!] \wedge [\![V := V + 1]\!]$$

That is,

$$(\exists x.\ (\overleftarrow{V} = x)\ \wedge\ \lceil\ \rceil\ \wedge\ (\overrightarrow{V} = x + 2))\ \wedge\ (\exists x.\ (\overleftarrow{V} = x)\ \wedge\ \lceil\ \rceil\ \wedge\ (\overrightarrow{V} = x + 1))$$

which is a contradiction. On the other hand, the semantics of these two consecutive assignments should be equal to the single assignment $(V := V + 3)$ by the merge assignment law [17]. Namely,

$$\exists x.\ (\overleftarrow{V} = x)\ \wedge\ \lceil\ \rceil\ \wedge\ (\overrightarrow{V} = x + 3)$$

Therefore a super-dense chop (written as $\bullet$) is introduced by [6] to denote the sequential composition (;). The super-dense chop assumes intermediate but invisible states to link together two operands of the operator. With $\bullet$, the two consecutive assignments

$$V =: V + 2; V =: V + 1$$

would be denoted by

$$[\![V =: V + 2]\!] \bullet [\![V := V + 1]\!]$$

that is,

$$(\exists x.\ (\overleftarrow{V} = x)\ \wedge\ \lceil\ \rceil\ \wedge\ (\overrightarrow{V} = x + 2)) \bullet (\exists x.\ (\overleftarrow{V} = x)\ \wedge\ \lceil\ \rceil\ \wedge\ (\overrightarrow{V} = x + 1)$$

For arbitrarily given initial value x of V, the intermediate but invisible state $(V = x + 2)$ can link the two operands of $\bullet$ in the above formula, and this formula can be simplified to

$$\exists x.\ (\overleftarrow{V} = x)\ \wedge\ \lceil\ \rceil\ \wedge\ (\overrightarrow{V} = x + 3)$$

which is the exact semantic formula of $(V := V + 3)$.

2 Higher-order Duration Calculus (HDC)

We will establish a higher-order duration calculus in this section, including its syntax, semantics, and proof system. In practice, we need to use the reals as time domain, so we will choose the reals as time model when we discuss its semantics in this section, i.e. $\mathbf{T}\text{ime} \mathrel{\widehat{=}} \mathbf{R}$. All intervals constructed by the reals defined as: $\mathbf{I}\text{ntv} \mathrel{\widehat{=}} \{[b,e] \mid b,e \in \mathbf{T}\text{ime} \wedge b \leq e\}$. Moreover, we will choose the reals as duration domains. Of course, we can define a much more general semantics of HDC, see Section 4.

2.1 Syntax and Semantics

Variables:

- *Global variables*, $GVar = \{x, y, z, x_1, y_1, z_1, ...\}$. They can be interpreted as values of real numbers (R).
- *Program variables*, $PVar = \{V, V_1, V_2, ...\}$. They are interpreted as finitely varied functions over time ($\mathsf{R} \ \rightarrow_{fv}\ \mathsf{R}$). As far as finite variability is concerned, we mean that infinite times of change of its value in a finite interval are excluded.
- *Length variable*, ℓ. It is a function from intervals to the reals ($\mathbf{Intv} \rightarrow \mathsf{R}$) which delivers the length of an interval.
- *Propositional letters*, $PLetter = \{X, X_1, X_2, ...\}$. They represent arbitrary formulas, and are interpreted as functions from intervals to truth values ($\mathbf{Intv} \rightarrow \{\mathbf{tt}, \mathbf{ff}\}$).

Function and predicate symbols

- *Function symbols* , $FSymb = \{f_i^n \mid i, n \geq 0\}$. If $n = 0$, f_i^n represents a constants. A function $f_i^n, n > 0$ is interpreted as an n-ary real total arithmetic function on $\mathbf{R}$. They are rigid, that is, their interpretation is independent of time and time intervals.
- *Predicate symbols*, $RSymb = \{R_i^n \mid i, n \geq 0\}$. If $n = 0$, R_i^n represents a Boolean constants, that is, **true** or **false**. If $n > 0$, the meaning of R_i^n is an n-ary Boolean valued total function on $\mathbf{R}$ which is independent of time and time intervals.

State terms:

A state term ϑ is a special term which is constructed from global variables and program variables by the following grammar:

$$\vartheta := \ x \mid V \mid f(\vartheta_1, \ldots, \vartheta_n)$$

Its meaning is a real valued function on time.

States:

Predicate of state terms are states, and if S_1 and S_2 are states, then $\neg S_1$, $S_1 \vee S_2$ and $S_1 \wedge S_2$ are also states. They are interpreted as a Boolean valued function on time domain ($\mathsf{R} \rightarrow \{0, 1\}$). E.g. $(V = x)$ and $(V > 3)$ both are states. Because we assume that program variables have finite variability, it is obviously that states have finite variability too. This is consistent with the assumption that state variables vary finitely in DC [1]. They can be constructed by the following grammar:

$$S := \ 0 \mid 1 \mid R(\vartheta_1, \ldots, \vartheta_n) \mid \neg S \mid S_1 \vee S_2$$

Terms:

We first introduce two special terms— initial and final values of state terms, $\overleftarrow{\vartheta}, \overrightarrow{\vartheta}, \overleftarrow{\vartheta_1}, \overrightarrow{\vartheta_1}, ...$, where $\leftarrow$ and $\rightarrow$ are functions with a domain of state terms and a range of functions from intervals to duration domains ($\mathbf{Intv} \rightarrow \mathsf{R}$). Given an interpretation of V in ($\mathsf{R} \rightarrow_{fv} \mathsf{R}$) and an interval $[b, e]$, $\overleftarrow{V} = d$ (or $\overrightarrow{V} = d$), iff there exists $\delta > 0$ such that the value of V is equal to d almost everywhere in the interval $[b - \delta, b]$ (or $[e, e + \delta]$ respectively). ($\overleftarrow{x} = d$) means ($x = d$) true in a left neighbourhood of the given interval, and therefore x is evaluated as d. Similarly, ($\overrightarrow{V_1 + V_2} = d$) means ($V_1 + V_2 = d$) true in a right neighbourhood of the given interval.

Terms are constructed as follows.

$$\theta ::= \; x \mid \ell \mid \; \overleftarrow{\theta} \; \mid \; \overrightarrow{\theta} \; \mid \textstyle\int S \mid f(\theta_1, \ldots, \theta_n)$$

where $\int S$ denotes the duration of S. That is, given interval $[b, e]$ and interpretation of S, S, the value of $\int S$ over $[b, e]$ is defined as

$$\textstyle\int_b^e \mathsf{S}(t)dt$$

where $\mathsf{S}(t) \in \{0, 1\}$.

Formulas:

Formulas are obtained from propositional letters and terms according to the following grammar.

$$\phi ::= \; X \mid R(\theta_1, \ldots, \theta_n) \mid \neg\phi \mid \phi \vee \phi \mid \phi\frown\phi \mid \exists x.\phi \mid \exists V.\phi$$

Under given interpretation of propositional letter, program variable and global variable, formulas are functions from intervals to truth values ($\mathbf{Intv} \rightarrow \{\mathbf{tt}, \mathbf{ff}\}$). For given interpretation and interval $[b, e]$, $\phi\frown\psi$ is true, iff there exists m ($b \leq m \leq e$) such that ϕ is true over $[b, m]$ for the given interpretation, and ψ is true over $[m, e]$ for the same interpretation. The semantics of $\exists x$ and $\exists V$ are conventional, and can be defined by changing the interpretation of x and V respectively.

A formula is rigid, if it does not contain ℓ, program variables, neither propositional letters. Similarly we can define rigid state terms, states and terms. When S is rigid, S can be lifted as a rigid formula. For example, $(x + y > 1)$ can be taken as a state as well as a formula according to the syntactic definitions above. In order to avoid confusion, when S is rigid, we will use ϕ_S to stand for the rigid formula corresponding to S.

A term θ is called *free for* x in ϕ if x does not occur freely in ϕ within a scope of $\exists x'$ or $\forall x'$ where x' is any variable occurring in θ.

2.2 Proof System

The proof system consists of four groups of axioms and rules.

Axioms and rules for interval temporal logic

The proof system still contains the inference system for ITL proposed by [9]. They are:

(IL1) $\quad \ell \geq 0$

(IL2) $\quad ((\phi\frown\psi) \wedge \neg(\phi\frown\varphi)) \;\Rightarrow\; (\phi\frown(\psi \wedge \neg\varphi))$
$\quad\quad ((\phi\frown\psi) \wedge \neg(\varphi\frown\psi)) \;\Rightarrow\; ((\phi \wedge \neg\varphi)\frown\psi))$

(IL3) $\quad ((\phi\frown\psi)\frown\varphi) \;\Leftrightarrow\; (\phi\frown(\psi\frown\varphi))$

(IL4) $\quad (\phi\frown\psi) \Rightarrow \; \phi \;$ if ϕ is rigid
$\quad\quad (\phi\frown\psi) \Rightarrow \; \psi \;$ if ψ is rigid

(IL5)	$((\exists x.\phi)^\frown\psi) \Rightarrow\ \exists x.\ \phi^\frown\psi$ if x is not free in ψ
	$(\phi^\frown\exists x.\psi) \Rightarrow\ \exists x.\ \phi^\frown\psi$ if x is not free in ϕ
(IL6)	$((\ell = x)^\frown\phi) \Rightarrow\ \neg((\ell = x)^\frown\neg\phi)$
	$(\phi^\frown(\ell = x)) \Rightarrow\ \neg(\neg\phi^\frown(\ell = x))$
(IL7)	$(x \geq 0 \wedge y \geq 0) \Rightarrow\ ((\ell = x + y) \Leftrightarrow\ ((\ell = x)^\frown(\ell = y)))$
(IL8)	$\phi \Rightarrow\ (\phi^\frown(\ell = 0))$
	$\phi \Rightarrow\ ((\ell = 0)^\frown\phi)$

The inference rules of ITL are:

$$\text{(N)}: \begin{array}{llll} \text{if} & \phi & \text{then} & \neg((\neg\phi)^\frown\psi) \\ \text{if} & \phi & \text{then} & \neg(\psi^\frown(\neg\phi)) \end{array}$$

$$\text{(M)}: \begin{array}{llll} \text{if} & \phi \Rightarrow \psi & \text{then} & (\phi^\frown\varphi) \Rightarrow (\psi^\frown\varphi) \\ \text{if} & \phi \Rightarrow \psi & \text{then} & (\varphi^\frown\phi) \Rightarrow (\varphi^\frown\psi) \end{array}$$

The axioms and rules of ITL also contain axioms and rules for propositional logic, predicate logic, and real arithmetic, such as

$$(\mathrm{G_x}): \quad \text{if } \phi \text{ then } \forall x.\phi$$

But we need to give a side condition for the following axiom.

$$(\mathrm{Q_x}): (\forall x.\phi(x)) \ \Rightarrow\ \phi(\theta) \begin{cases} \text{if either } r \text{ is free for } x \text{ in } \phi(x) \text{ and } r \text{ is rigid} \\ \text{or } r \text{ is free for } x \text{ in } \phi(x) \text{ and } \phi(x) \text{ is chop free.} \end{cases}$$

The axioms and rules for classical logic will not be listed here, but can be found in [18]. We will use PL to stand for all axioms and rules for propositional logic, predicate logic, and real arithmetic later.

The following theorem which will be referred later can be derived from the above.

Theorem 1

(I)	$\phi \Rightarrow \Box\phi$ *if* ϕ *is rigid*	(III)	$\exists x.\ell = x$
(II)	$\phi^\frown(\ell = 0) \Leftrightarrow \phi$ $(\ell = 0)^\frown\phi \Leftrightarrow \phi$	(IV)	$\ell = 0 \Rightarrow \Box(\ell = 0)$
		(V)	$(\phi \vee \psi)^\frown\varphi \Leftrightarrow ((\phi^\frown\varphi) \vee (\psi^\frown\varphi))$

Axioms and rules for durations

The axioms for durations include:

(DC1)	$\int 0 = 0$	(DC4)	$\int S_1 + \int S_2 = \int(S_1 \vee S_2) + \int(S_1 \wedge S_2)$
(DC2)	$\int 1 = \ell$	(DC5)	$((\int S = x_1)^\frown(\int S = x_2)) \Rightarrow (\int S = x_1 + x_2)$
(DC3)	$\int S \geq 0$	(DC6)	$\int S_1 = \int S_2$, if $S_1 \Leftrightarrow S_2$
		(DC7)	$\lceil S \rceil \Leftrightarrow (\phi_S \wedge \ell > 0)$, if S is rigid

According to the above axioms and rules, we can prove the following properties about $\lceil S \rceil$.

Theorem 2

(I)	$\lceil 1 \rceil \vee \lceil\ \rceil$
(II)	$\int S \leq \ell$
(III)	$\lceil S \rceil \vee \lceil\ \rceil \ \Rightarrow\ \Box(\lceil S \rceil \vee \lceil\ \rceil)$

Axioms and rules for program variables

In order to define program variables as finitely varied functions, we introduce two induction rules, where $H(X)$ is a formula containing the propositional letter X.

$$\begin{array}{llllll}
\text{(DCR1)} & \text{if} & H(\lceil\,\rceil) & \text{and} & H(X) \Rightarrow H(X \vee (X ^\frown \exists x.\lceil V = x \rceil)), & \text{then } H(\mathbf{true}) \\
\text{(DCR2)} & \text{if} & H(\lceil\,\rceil) & \text{and} & H(X) \Rightarrow H(X \vee ((\exists x.\lceil V = x \rceil) ^\frown X)), & \text{then } H(\mathbf{true})
\end{array}$$

$\lceil V = x \rceil$ means that program variable V keeps value x almost everywhere in a non-point interval. Therefore, these two induction rules stipulate that any interval can be covered by consecutive subintervals, where V keeps constant. This is exactly what is meant by the finite variability of program variables.

Remark: The above two induction rules are powerful enough to ensure the proof system is relatively complete as in [15], see [8]. [9] has shown that finite variability cannot be axiomatised by finite rules on abstract domains, so the two rules are not efficient enough to axiomatise finite variability of program variables on abstract domains. [16] introduced a ω -rule into ITL in order to axiomatise finite variability on abstract domains. Therefore, we will use a ω -rule instead of the two rules when to discuss the completeness of the proof system on abstract domains in Section 4. Besides, we can show that the ω -rule is stronger than DCR1 and DCR2, so all results given below are still valid after replacing DCR1 and DCR2 with the ω -rule.

For the initial and final values of ϑ, $\overleftarrow{\vartheta}$ and $\overrightarrow{\vartheta}$, we establish the following axioms.

$$\begin{array}{lcl}
\text{(PV1)} & (\ell > 0)^\frown((\overleftarrow{\vartheta} = x_1) \wedge (\ell = x_2)) & \Leftrightarrow\ \mathbf{true}^\frown\lceil \vartheta = x_1 \rceil^\frown(\ell = x_2) \\
\text{(PV2)} & ((\overrightarrow{\vartheta} = x_1) \wedge (\ell = x_2))^\frown(\ell > 0) & \Leftrightarrow\ (\ell = x_2)^\frown\lceil \vartheta = x_1 \rceil^\frown\mathbf{true}
\end{array}$$

PV1 and PV2 formulate the meaning of the initial value and final value of a state term which are inherited from the previous segment of a program, and passed to the successive one. Because the function $\leftarrow$ ($\rightarrow$) involves the value of a state term at left neighbourhood (right neighbourhood), the neighbourhood rule [11] is necessary in order to reason about the properties for them.

NR if $(\ell = a)^\frown\Psi^\frown(\ell = b) \Rightarrow (\ell = a)^\frown\Upsilon^\frown(\ell = b)$, then $\Psi \Rightarrow \Upsilon$. where, $(a, b \geq 0)$

Using NR, we can prove the following useful theorem.

Theorem 3

$$\begin{array}{ll}
\text{(I)} & (\overleftarrow{V} = x)^\frown(\ell = y) \ \Leftrightarrow\ (\overleftarrow{V} = x) \wedge (\ell \geq y) \\
\text{(II)} & (\ell = y)^\frown(\overrightarrow{V} = x) \ \Leftrightarrow\ (\overrightarrow{V} = x) \wedge (\ell \geq y) \\
\text{(III)} & (\overleftarrow{\vartheta} = \vartheta) \wedge (\overrightarrow{\vartheta} = \vartheta) \quad \textit{if } \vartheta \textit{ is rigid} \\
\text{(IV)} & (\overleftarrow{f(\vartheta_1, \ldots, \vartheta_n)} = f(\overleftarrow{\vartheta_1}, \ldots, \overleftarrow{\vartheta_n})) \quad \textit{where } f \in Fsymb \\
\text{(V)} & (\overrightarrow{f(\vartheta_1, \ldots, \vartheta_n)} = f(\overrightarrow{\vartheta_1}, \ldots, \overrightarrow{\vartheta_n})) \quad \textit{where } f \in Fsymb
\end{array}$$

In the theorem, (I) and (II) specify that an interval shares left (right) neighbourhood with its prefix (suffix) subintervals. (III), (IV) and (V) tell how to calculate the initial value and final value of a composite term.

Formulas are properties of program variables and their initial and final values. Let us assume that ϕ includes V, $\overleftarrow{V}$ and $\overrightarrow{V}$, and ψ includes only V but not $\overleftarrow{V}$ nor $\overrightarrow{V}$. Then we can prove

Theorem 4

$$\begin{array}{ll}
\text{(I)} & (\lceil V_1 = V_2 \rceil \vee \lceil\,\rceil) \wedge (\overleftarrow{V_1} = \overleftarrow{V_2}) \wedge (\overrightarrow{V_1} = \overrightarrow{V_2}) \ \Rightarrow\ \phi(V_1) \Leftrightarrow \phi(V_2) \\
\text{(II)} & \lceil V_1 = V_2 \rceil \vee \lceil\,\rceil \ \Rightarrow\ \psi(V_1) \Leftrightarrow \psi(V_2)
\end{array}$$

Axioms and rules for higher-order quantifications

Finally, we will provide a group of axioms and rules to specify the semantics of V, $\overleftarrow{V}$ and $\overrightarrow{V}$ in the context of quantifications.

The axiom and rule below are standard as in predicate logic.

$$\begin{array}{ll} \mathrm{G_V}: & \text{if} \quad \phi \quad \text{then} \quad \forall V.\phi \\ \mathrm{Q_V}: & (\forall V.\phi(V)) \ \Rightarrow \ \phi(\vartheta) \end{array}$$

The following axioms say that if $\overleftarrow{V}$ is not defined by formula ϕ, then it can take any value, since the value of $\overleftarrow{V}$ is defined by value of V outside the reference interval with respect to ϕ. Similarly for $\overrightarrow{V}$.

$$\begin{array}{lll} \text{(HDC1)} & \exists V.\phi \ \Rightarrow \ \exists V.\ \phi \wedge (\overleftarrow{V} = x) & \text{if } \overleftarrow{V} \notin \phi \\ \text{(HDC2)} & \exists V.\phi \ \Rightarrow \ \exists V.\ \phi \wedge (\overrightarrow{V} = x) & \text{if } \overrightarrow{V} \notin \phi \end{array}$$

The distributivity of $\exists V$ over the chop operator is the most essential property of V as a function over time. $\exists V$ can distribute over the chop, if and only if the value of $\overrightarrow{V}$ in the left operand of the chop can match the value of V in the right operand, and symmetrically for the value of $\overleftarrow{V}$ in the right operand.

$$\text{(HDC3)} \qquad \left(\begin{array}{ll} & (\exists V.\ \phi \wedge (\mathbf{true}^\frown \lceil\!\lceil V = x_1 \rceil\!\rceil \vee \lceil\!\lceil\rceil\!\rceil) \wedge (\overrightarrow{V} = x_2)) \\ ^\frown & (\exists V.\ \psi \wedge (\lceil\!\lceil V = x_2 \rceil\!\rceil^\frown \mathbf{true} \vee \lceil\!\lceil\rceil\!\rceil) \wedge (\overleftarrow{V} = x_1)) \end{array} \right) \Rightarrow \ \exists V.\ \phi^\frown \psi$$

We can show the following theorem about the distributivity of $\exists V$ over the chop operator.

Theorem 5

$$\begin{array}{lll} \text{(I)} & (\exists V.\phi)^\frown \psi \Leftrightarrow \exists V.(\phi^\frown \psi) & \textit{if } V \notin \psi \\ \text{(II)} & \phi^\frown \exists V.\psi \Leftrightarrow \exists V.(\phi^\frown \psi) & \textit{if } V \notin \phi \end{array}$$

3 Real-time Semantics of Programs

In this section, we will explain how to apply HDC to define semantics for real-time programs. The method provided in the paper can also be applied to deal with the behaviour of communications, e.g. we can define a semantics of declaring internal channel just as for local variable declaration.

3.1 Super-dense Computation

It is a comfortable abstraction to assume that a sequence of program statements is executed one by one, but consumes no time by comparison with an environment of a grand time granularity. A computation of a sequence of operations which is assumed timeless is called a super-dense computation [5]. In order to express super-dense computation of program in the framework of DC, [6] introduces the super-dense chop $\bullet$. With $\bullet$, we can link two formulas with an intermediate but invisible state. So, for initial value x of V, with $(V = x + 2)$ as the intermediate state, the formula

$$(\exists x.\ (\overleftarrow{V} = x) \ \wedge \ \lceil\!\lceil\ \rceil\!\rceil \ \wedge \ (\overrightarrow{V} = x + 2)) \bullet (\exists x.\ (\overleftarrow{V} = x) \ \wedge \ \lceil\!\lceil\ \rceil\!\rceil \ \wedge \ (\overrightarrow{V} = x + 1))$$

will become

$$\exists x.\ (\overleftarrow{V}= x)\ \wedge\ \lceil\ \rceil\ \wedge\ (\overrightarrow{V}= x+3)$$

It is indicated in [4] that the super-dense chop can be derived from the quantifications over program variables. Let us assume that $\phi(V_1,...,V_n)$ and $\psi(V_1,...,V_n)$ have $V_1,...,V_n$ as their only free program variables. We define the super-dense chop by

$$\begin{array}{ll} & \phi(V_1,...,V_n)\bullet\psi(V_1,...,V_n)\\ \widehat{=} & \exists x_1,...,x_n,V_{11},...,V_{1n},V_{21},...,V_{2n}.\\ & (\phi(V_{11},...,V_{1n})\wedge_{i=1}^{n}((\lceil V_i=V_{1i}\rceil\vee\lceil\ \rceil)\wedge(\overleftarrow{V_i}=\overleftarrow{V_{1i}})\wedge(\overrightarrow{V_{1i}}=x_i)))\\ & {}^\frown(\psi(V_{21},...,V_{2n})\wedge_{i=1}^{n}((\lceil V_i=V_{2i}\rceil\vee\lceil\ \rceil)\wedge(\overrightarrow{V_i}=\overrightarrow{V_{2i}})\wedge(\overleftarrow{V_{2i}}=x_i)))\end{array}$$

The right hand side of the definition stipulates that we can use $(V_i = x_i)$ $(i = 1,...,n)$ as intermediate but invisible states to link ϕ and ψ together. In other words, the value x_i of V_i is instantly passed to ψ from ϕ through the invisible state. Therefore, the semantics of the sequential composition operator of programs can be defined as

$$[\![\mathcal{P}_1;\mathcal{P}_2]\!]\ \widehat{=}\ [\![\mathcal{P}_1]\!]\bullet[\![\mathcal{P}_2]\!]$$

By Theorem 3 (I)&(II), HDC1 and HDC2, we can establish a theorem about passing value of program variable through sequential composition.

Theorem 6 *If* $\leftarrow,\rightarrow\notin\phi,\psi$, *then*

$$\begin{array}{ll} & ((\overleftarrow{V}=x_1)\wedge\phi\wedge(\overrightarrow{V}=x_2))\bullet((\overleftarrow{V}=x_3)\wedge\psi\wedge(\overrightarrow{V}=x_4))\\ \Leftrightarrow & (\phi^\frown\psi)\wedge(\overleftarrow{V}=x_1)\wedge(\overrightarrow{V}=x_4)\wedge\ (x_2=x_3)\end{array}$$

Proof: Let us assume that V is the only free program variable of ϕ and ψ. We first prove the first half of $\Leftrightarrow$.

$$\begin{array}{lll} & (\overleftarrow{V}=x_1)\wedge\phi\wedge(\overrightarrow{V}=x_2))\bullet((\overleftarrow{V}=x_3)\wedge\psi\wedge(\overrightarrow{V}=x_4)) & \\ \Rightarrow & \exists x,V_1,V_2.((\overleftarrow{V_1}=x_1)\wedge\phi[V_1/V]\wedge(\overrightarrow{V_1}=x_2) & \\ & \wedge(\lceil V=V_1\rceil\vee\lceil\ \rceil)\wedge(\overleftarrow{V}=\overleftarrow{V_1})\wedge(\overrightarrow{V_1}=x))^\frown((\overleftarrow{V_2}=x_3) & \\ & \wedge\psi[V_2/V]\wedge(\overrightarrow{V_2}=x_4)\wedge(\lceil V=V_2\rceil\vee\lceil\ \rceil)\wedge(\overrightarrow{V}=\overrightarrow{V_2})\wedge(\overleftarrow{V_2}=x)) & (\text{Def }\bullet)\\ \Rightarrow & \exists x,V_1,V_2.(\phi[V_1/V]\wedge(\lceil V=V_1\rceil\vee\lceil\ \rceil)\wedge(\overleftarrow{V}=\overleftarrow{V_1}=x_1)\wedge(\overrightarrow{V_1}=x=x_2)) & \\ & {}^\frown(\psi[V_2/V]\wedge(\lceil V=V_2\rceil\vee\lceil\ \rceil)\wedge(\overrightarrow{V}=\overrightarrow{V_2}=x_4)\wedge(\overleftarrow{V_2}=x=x_3)) & (\text{PL})\\ \Rightarrow & (\overleftarrow{V}=x_1)\wedge(\overrightarrow{V}=x_4)\wedge(x_2=x_3) & (\text{Theorem 3 (I)}\\ & \wedge\exists V_1,V_2.(\phi[V_1/V]\wedge(\lceil V=V_1\rceil\vee\lceil\ \rceil))^\frown(\psi[V_2/V]\wedge(\lceil V=V_2\rceil\vee\lceil\ \rceil)) & \&(\text{II}),\text{ IL4})\\ \Rightarrow & (\overleftarrow{V}=x_1)\wedge(\overrightarrow{V}=x_4)\wedge\ (x_2=x_3)\wedge\exists V_1,V_2.\ (\phi^\frown\psi) & (\text{Theorem 4 (II)})\\ \Rightarrow & (\overleftarrow{V}=x_1)\wedge(\overrightarrow{V}=x_4)\wedge\ (x_2=x_3)\wedge(\phi^\frown\psi) & (\text{PL})\end{array}$$

The other half can be proved as:

$$
\begin{array}{lll}
 & (\phi^\frown\psi) \land (\overleftarrow{V}= x_1) \land (\overrightarrow{V}= x_4) \land \ (x_2 = x_3) & \\
\Rightarrow & (x_2 = x_3) \land ((\phi \land (\overleftarrow{V}= x_1))^\frown(\psi \land (\overrightarrow{V}= x_4))) & \text{(Theorem 3 (I)\&(II))} \\
\Rightarrow & (x_2 = x_3) \ \land ((\overleftarrow{V}= x_1) \land \phi \land (\lceil V = V\rceil \lor \lceil\ \rceil) \land (\overleftarrow{V}=\overleftarrow{V})) & \\
 & {}^\frown(\psi \land (\overrightarrow{V}= x_4) \land (\lceil V = V\rceil \lor \lceil\ \rceil) \land (\overrightarrow{V}=\overrightarrow{V})) & \text{(Theorem 2 (I))} \\
\Rightarrow & (x_2 = x_3) \ \land \ (\exists V_1.(\overleftarrow{V_1}= x_1) \land \phi[V_1/V] \land (\lceil V = V_1\rceil \lor \lceil\ \rceil) \land (\overleftarrow{V}=\overleftarrow{V_1})) & \\
 & {}^\frown(\exists V_2.\psi[V_2/V] \land (\overrightarrow{V_2}= x_4) \land (\lceil V = V_2\rceil \lor \lceil\ \rceil) \land (\overrightarrow{V}=\overrightarrow{V_2})) & (\mathrm{Q_V}) \\
\Rightarrow & (x_2 = x_3) & \\
 & \land(\exists V_1.(\overleftarrow{V_1}= x_1) \land \phi[V_1/V] \land (\overleftarrow{V}=\overleftarrow{V_1}) \land (\lceil V = V_1\rceil \lor \lceil\ \rceil) \land (\overrightarrow{V_1}= x_2)) & \\
 & {}^\frown(\exists V_2.\psi[V_2/V] \land (\overrightarrow{V_2}= x_4) \land (\overrightarrow{V}=\overrightarrow{V_2}) \land (\lceil V = V_2\rceil \lor \lceil\ \rceil) \land (\overleftarrow{V_2}= x_3)) & \text{(HDC1\&2)} \\
\Rightarrow & \exists x, V_1, V_2.((\overleftarrow{V_1}= x_1) \land \phi[V_1/V] \land (\overrightarrow{V_1}= x_2) \land (\lceil V = V_1\rceil \lor \lceil\ \rceil) & \\
 & \land(\overleftarrow{V}=\overleftarrow{V_1}) \land (\overrightarrow{V_1}= x))^\frown((\overleftarrow{V_2}= x_3) \land \psi[V_2/V] \land (\overrightarrow{V_2}= x_4) & \\
 & \land(\lceil V = V_2\rceil \lor \lceil\ \rceil) \land (\overrightarrow{V}=\overrightarrow{V_2}) \land (\overleftarrow{V_2}= x)) & \text{(PL)} \\
\Rightarrow & ((\overleftarrow{V}= x_1) \land \phi \land (\overrightarrow{V}= x_2)) \bullet ((\overleftarrow{V}= x_3) \land \psi \land (\overrightarrow{V}= x_4)) & (\text{Def } \bullet)
\end{array}
$$

A corollary of Theorem 6 is that if $\leftharpoonup, \rightharpoonup \notin \phi, \psi$, then

$$\phi \bullet \psi \ \Leftrightarrow \ \phi^\frown\psi$$

By Theorem 6, it is easy to prove the example of the two consecutive assignments given in the introduction satisfies the emerge assignment law.

$$[\![V := V + 2; V := V + 1]\!] \ \Leftrightarrow \ [\![V := V + 3]\!]$$

We can show that the sequential composition of programs has many nice properties. For examples, it is associative and takes skip as unit.

3.2 Semantics of Local Declaration

Declaring a local variable, V, semantically corresponds to the existential quantification of V. The semantics of (begin V:$\mathcal{P}$ end) where V is declared as a local variable should be

$$\exists V.\ [\![\mathcal{P}]\!]$$

where $[\![\mathcal{P}]\!]$ stands for the semantics of $\mathcal{P}$.

Now, we consider an example of a procedure of a delay of k time units as (begin V:$\mathcal{Q}(V, k)$ end), where $\mathcal{Q}(V, k)$ is

$$V := 0;\ \textsf{while}\ V < k\ \textsf{do}\ (\textsf{tick};\ V := V + 1)$$

k is a positive integer, and tick is taken as a one-unit delay. Let wait(k) stand for the procedure. We now define the semantics of this procedure. tick is a one-unit delay, and will not change the value of the local variable V. Thus,

$$[\![\textsf{tick}]\!] \ \widehat{=} \ \exists x.\ (\overleftarrow{V}= x) \land \lceil V = x\rceil \land (\ell = 1) \land (\overrightarrow{V}= x)$$

and the semantics of the while body is

$$\begin{array}{ll} & [\![\text{tick}; V := V+1]\!] \\ \widehat{=} & (\exists x.\ (\overleftarrow{V}= x) \wedge \lceil V = x \rceil \wedge (\ell = 1) \wedge (\overrightarrow{V}= x)) \bullet (\exists x.\ (\overleftarrow{V}= x) \wedge \lceil\ \rceil \wedge (\overrightarrow{V}= x+1)) \\ \Leftrightarrow & \exists x.\ (\overleftarrow{V}= x) \wedge \lceil V = x \rceil \wedge (\ell = 1) \wedge (\overrightarrow{V}= x+1) \quad \text{(Theorem 6)} \end{array}$$

$(V := 0)$ is to receive an initial value of V and pass 0 as final value of V instantly. Namely

$$[\![V := 0]\!] \ \widehat{=}\ \exists x.\ (\overleftarrow{V}= x) \wedge \lceil\ \rceil \wedge (\overrightarrow{V}= 0)$$

The while is to execute its body repeatedly until its Boolean condition is violated. The execution of the body may go on infinitely, and a divergence happens. However, in the case of $\mathcal{Q}(V,k)$, the while body will be only executed for k times, given the initial value of V as 0. Therefore,

$$\begin{array}{lll} [\![\mathsf{wait}(k)]\!] & \widehat{=} & \exists V.\ [\![V := 0]\!] \bullet [\![\mathsf{while}]\!] \\ & \Leftrightarrow & \exists V. (\exists x.\ (\overleftarrow{V}= x) \wedge \lceil\ \rceil \wedge (\overrightarrow{V}= 0)) \\ & & \bullet (\exists x < k.\ (\overleftarrow{V}= x) \wedge \lceil V = x \rceil \wedge (\ell = 1) \wedge (\overrightarrow{V}= x+1))^k \end{array}$$

where,

$$\phi^0 \ \widehat{=}\ I \qquad\qquad \phi^{n+1} \ \widehat{=}\ \phi \bullet \phi^n$$

Using Theorem 6, HDC2 and HDC3, we can simplify the above formula, and have

$$[\![\mathsf{wait}(k)]\!] \ \Leftrightarrow\ (\ell = k)$$

When we call this procedure with 3 as its actual parameter in a program environment where V is as the only global program variable, the procedure call will postpone the execution of the program for 3 time units while the value of V remains. That is,

$$\begin{array}{lll} [\![\mathsf{call\ wait}(3)]\!] & \widehat{=} & [\![\mathsf{wait}(3)]\!] \wedge \exists x.\ (\overleftarrow{V}= x) \wedge (\lceil V = x \rceil \vee \lceil\ \rceil) \wedge (\overrightarrow{V}= x) \\ & \Leftrightarrow & \exists x.\ (\overleftarrow{V}= x) \wedge \lceil V = x \rceil \wedge (\ell = 3) \wedge (\overrightarrow{V}= x) \end{array}$$

3.3 Decomposition of Real-time Semantics

It is desirable to decompose the real-time semantics of a program into two separate parts, untimed part and timed part. In the example of $[\![\mathsf{call\ wait}(3)]\!]$, its semantics can be written as

$$\exists x.\ (\overleftarrow{V}=\overrightarrow{V}= x) \wedge (\lceil V = x \rceil \wedge (\ell = 3))$$

where $(\overleftarrow{V}=\overrightarrow{V}= x)$ is a rigid relation of initial and final values of program variable V which represents the untimed behaviour of the procedure call, and $(\lceil V = x \rceil \wedge (\ell = 3))$ represents the timed behaviour of the procedure call, including stable state $(V = x)$ and its stable period $(\ell = 3)$. A rigid variable x links these two parts to define the relation of initial, final and stable values of V.

With this decomposition, we can regard the real-time semantics as a conservative extension of untimed one, and hence many existing techniques for untimed program behaviour can still be applied, when we synthesise and analyse real-time program behaviour.

In [4], a theorem about the decomposition is presented and proved. The following theorem generalises the above one, and demonstrates the feasibility of the decomposition. Let us assume that

$$\exists x_1, ..., x_n.\ \phi_1 \wedge G_1(\overleftarrow{V}, \overrightarrow{V}) \quad \text{and} \quad \exists y_1, ..., y_n.\ \phi_2 \wedge G_2(\overleftarrow{V}, \overrightarrow{V})$$

are semantics of two program segments with V as only free program variable. ϕ_i $(i = 1, 2)$ represent the timed behaviour and do not contains any $\leftarrow$ and $\rightarrow$. $G_i(x, y)$ $(i = 1, 2)$ are rigid formulas, and $G_i(\overleftarrow{V}, \overrightarrow{V})$ $(i = 1, 2)$ specify the untimed behaviour of these two segments. $x_1, ..., x_n$ and $y_1, ..., y_n$ be rigid variables to link the untimed and timed behaviour of these two segments respectively. We now prove that the sequential operator maintains the decomposition.

Theorem 7

$$\begin{array}{ll} & (\exists x_1, ..., x_n.\ \phi_1 \wedge G_1(\overleftarrow{V}, \overrightarrow{V})) \bullet (\exists y_1, ..., y_n.\ \phi_2 \wedge G_2(\overleftarrow{V}, \overrightarrow{V})) \\ \Leftrightarrow & \exists x_1, ..., x_n, y_1, .., , y_n.\ \phi_1 \frown \phi_2 \ \wedge\ \exists x.\ G_1(\overleftarrow{V}, x) \wedge G_2(x, \overrightarrow{V}) \end{array}$$

Proof:

$$\begin{array}{lll} & (\exists x_1, ..., x_n.\ \phi_1 \wedge G_1(\overleftarrow{V}, \overrightarrow{V})) \bullet (\exists y_1, ..., y_n.\ \phi_2 \wedge G_2(\overleftarrow{V}, \overrightarrow{V})) & \\ \Leftrightarrow & \exists x_1, ..., x_n, y_1, ..., y_n, z_1, ..., z_4. & \\ & (\phi_1 \wedge G_1(z_1, z_2) \wedge (\overleftarrow{V} = z_1) \wedge (\overrightarrow{V} = z_2)) & \\ & \bullet(\phi_2 \wedge G_2(z_3, z_4) \wedge (\overleftarrow{V} = z_3) \wedge (\overrightarrow{V} = z_4)) & \text{(PL)} \\ \Leftrightarrow & \exists x_1, ..., x_n, y_1, ..., y_n, z_1, ..., z_4. & \\ & (\phi_1 \frown \phi_2) \wedge G_1(z_1, z_2) \wedge (\overleftarrow{V} = z_1) \wedge G_2(z_3, z_4) \wedge (\overrightarrow{V} = z_4) \wedge (z_2 = z_3) & \text{(Theorem 6, IL4)} \\ \Leftrightarrow & \exists x_1, ..., x_n, y_1, ..., y_n.\ (\phi_1 \frown \phi_2) \wedge \exists x.\ G_1(\overleftarrow{V}, x) \wedge G_2(x, \overrightarrow{V}) & \text{(PL)} \end{array}$$

4 Completeness of HDC on Abstract Domains

The above discussion is based on taking time domain and duration domain as the reals, and it is indeed needed in practice. But as indicated in the introduction, any system which takes the reals as its only model is not complete. Hence, we shall study completeness of HDC based on a much more general semantics in this section, that is, we shall prove its completeness on abstract domains which satisfies the axioms for a totally ordered commutative group. Therefore the following completeness result seems the best one we can expect for HDC.

4.1 The Semantics of HDC on Abstract Domains

We first give a semantics of HDC on abstract domains.

Definition 1 *A* time domain *is a linearly ordered set* $< T, \leq >$.

Definition 2 *Given a time domain* $< T, \leq >$, *we can define a set of intervals* $\mathbf{I}ntv\,(T) = \{[t_1, t_2] \mid t_1, t_2 \in T \text{ and } t_1 \leq t_2\}$, *where* $[t_1, t_2] = \{t \mid t \in T \text{ and } t_1 \leq t \leq t_2\}$.

Definition 3 *A* duration domain *is a system of the type* $< D, +, 0 >$, *which satisfies the following axioms:*

$$\begin{array}{ll} (D1) & a + (b + c) = (a + b) + c \\ (D2) & a + 0 = a = 0 + a \\ (D3) & a + b = a + c \Rightarrow b = c, \quad a + c = b + c \Rightarrow a = b \\ (D4) & a + b = 0 \Rightarrow a = 0 = b \\ (D5) & \exists c.a + c = b \vee b + c = a, \exists c.c + a = b \vee c + b = a \end{array}$$

I.e. $< D, +, 0 >$ *is a totally ordered commutative group.*

Definition 4 *Given a time domain $< T, \leq >$ and a duration domain $< D, +, 0 >$, a measure m is a function from T to D which satisfies the following conditions:*

$$\begin{array}{ll} (M1) & m([t_1, t_2]) = m([t_1, t_2']) \Rightarrow t_2 = t_2' \\ (M2) & m([t_1, t]) + m([t, t_2]) = m([t_1, t_2]) \\ (M3) & m([t_1, t_2]) = a + b \Rightarrow \exists t. m([t_1, t]) = a \wedge (t_1 \leq t \leq t_2) \end{array}$$

Definition 5 *A frame of HDC is a triple of $<< T, \leq >, < D, +, 0 >, m >$, where $< T, \leq >$ is a time domain, $< D, +, 0 >$ is a duration domain, m is a measure. We will use $\mathcal{F}$ to denote frames.*

Definition 6 *A model of HDC is a quadruple $<< T, \leq >, < D, +, 0 >, m, \mathcal{I} >$, where, $<< T, \leq >, < D, +, 0 >, m >$ is a frame, $\mathcal{I}$ is an interpretation, which satisfies the following conditions:*

(I) $\mathcal{I}(X) : \ \mathbf{I}ntv\,(T) \rightarrow \{0, 1\}$ *for all* $X \in PLetter$*;*

(II) $\mathcal{I}(R_i^n) : \ D^n \rightarrow \{0, 1\}$ *for all* $R_i^n \in RSymb$*;*

(III) $\mathcal{I}(f_i^n) : D^n \rightarrow D$ *for all* $f_i^n \in FSymb$*;*

(IV) *For any* $V \in PVar$ *and interval* $[t_1, t_2] \in \mathbf{I}ntv(T)$*, there exists a sequence* $t_1', \ldots, t_n'$ *such that* $t_1 = t_1' \leq \ldots \leq t_n' = t_2$*, and for any* $t, t' \in [t_i', t_{i+1}')$*, we have* $\mathcal{I}(V)(t) = \mathcal{I}(V)(t')$*;*

(V) $\mathcal{I}(0) = 0, \mathcal{I}(+) = +, \mathcal{I}(=)$ *is* $=$*, and* $\mathcal{I}(\ell) = m$

The above condition (IV) stipulates that for any interpretation, the meaning of program variables under it has the finite variability.

Definition 7 *Let $\mathcal{I}$ and $\mathcal{I}'$ be two interpretations that satisfies the above conditions. $\mathcal{I}$ is z-equivalent to $\mathcal{I}'$ if $\mathcal{I}$ and $\mathcal{I}'$ have same values to all symbols, but possibly z.*

The semantics of a state term ϑ, given a model $\mathcal{M} =< \mathcal{F}, \mathcal{I} >$, is a function:

$$\mathcal{I}(\vartheta) \in T \ \rightarrow \ D$$

defined inductively on its structure as follows:

$$\begin{array}{lcl} \mathcal{I}(x)(t) & = & \mathcal{I}(x) \\ \mathcal{I}(V)(t) & = & \mathcal{I}(V)(t) \\ \mathcal{I}(f^n(\vartheta_1, \ldots, \vartheta_n))(t) & = & \mathcal{I}(f^n)(\mathcal{I}(\vartheta_1)(t), \ldots, \mathcal{I}(\vartheta_n)(t)) \end{array}$$

The semantics of a state expression S, given a model $\mathcal{M} =< \mathcal{F}, \mathcal{I} >$, is a function:

$$\mathcal{I}(S) \in T \ \rightarrow \ \{0, 1\}$$

defined inductively on its structure as follows:

$$\begin{array}{lcl} \mathcal{I}(0)(t) & = & 0 \\ \mathcal{I}(1)(t) & = & 1 \\ \mathcal{I}(R^n(\vartheta_1, \ldots, \vartheta_n))(t) & = & \mathcal{I}(R^n)(\mathcal{I}(\vartheta_1)(t), \ldots, \mathcal{I}(\vartheta_n)(t)) \\ \mathcal{I}(\neg S)(t) & = & 1 - \mathcal{I}(S)(t) \\ \mathcal{I}(S_1 \vee S_2)(t) & = & \left\{ \begin{array}{l} 0 \text{ if } \mathcal{I}(S_1)(t) = 0 \text{ and } \mathcal{I}(S_2)(t) = 0 \\ 1 \text{ otherwise} \end{array} \right. \end{array}$$

Lemma 1 *Let S be a state expression and $\mathcal{I}$ be an interpretation of the symbols in HDC on a frame $\mathcal{F} =<< T, \leq>, < D, +, 0 >, m >$. Then for every $[t_1, t_2] \in \mathbf{I}ntv(T)$ there exist $t'_1, \ldots, t'_n$ such that $t_1 = t'_1 \leq \ldots \leq t'_n = t_2$, and for any $t, t' \in [t'_i, t'_{i+1})$ implies $\mathcal{I}(S)(t) = \mathcal{I}(S)(t')$ for all $i = 1, \ldots, n-1$.*

Proof: Induction on the construction of S. $\square$

Using Lemma 1, we can give the interpretation of $\int S$ under a HDC model $\mathcal{M} =<< T, \leq>, < D, +, 0 >, m, \mathcal{I} >$ as follows: Let $[t_1, t_2] \in \mathbf{I}\text{ntv}(T)$. Let $t'_1, \ldots, t'_n$ have the property stated in Lemma 1. Let us define $p \bullet c$ for $p \in \{0, 1\}$ and $c \in D$ as follows:

$$p \bullet c \quad = \quad \begin{cases} 0 & \text{if } p = 0 \\ c & \text{if } p = 1 \end{cases}$$

Then $\mathcal{I}(\int S)([t_1, t_2]) = \sum_{i=1}^{n-1} \mathcal{I}(S)(t'_i) \bullet m([t'_i, t'_{i+1}])$. It is easy to show that this definition does not depend on the particular choice $t'_1, \ldots, t'_n$.

Given a model $\mathcal{M} =<< T, \leq>, < D, +, 0 >, m, \mathcal{I} >$, and an interval $[t_1, t_2] \in \mathbf{I}\text{ntv}(T)$, the meaning of initial and final values of state terms $\overleftarrow{\vartheta}, \overrightarrow{\vartheta}, \overleftarrow{\vartheta_1}, \overrightarrow{\vartheta_1}, \ldots$, are functions $\mathbf{I}\text{ntv}(T) \rightarrow D$ defined as follows:

$\mathcal{I}(\overleftarrow{\vartheta}, [t_1, t_2]) = d$,
iff $< \mathcal{F}, \mathcal{I} >, [t_1 - \delta, t_1] \vdash_{HDC} \lceil \vartheta = d \rceil$, for some $\delta > 0$.

$\mathcal{I}(\overrightarrow{\vartheta}, [t_1, t_2]) = d$,
iff $< \mathcal{F}, \mathcal{I} >, [t_2, t_2 + \delta] \vdash_{HDC} \lceil \vartheta = d \rceil$, for some $\delta > 0$.

where,

$$\begin{array}{lcl} \mathcal{M}, [t_1, t_2] \models_{HDC} \phi & \widehat{=} & \mathcal{I}(\phi)([t_1, t_2]) = tt \\ \mathcal{M}, [t_1, t_2] \not\models_{HDC} \phi & \widehat{=} & \mathcal{I}(\phi)([t_1, t_2]) = ff \end{array}$$

The meaning of a formula ϕ, Given a HDC model $\mathcal{M} =<< T, \leq>, < D, +, 0 >, m, \mathcal{I} >$, is a function:

$$\mathcal{I}(\phi) \in \mathbf{I}\text{ntv}(T) \rightarrow \{tt, ff\}$$

defined inductively on its structures as follows:

1. $\mathcal{M}, [t_1, t_2] \models_{HDC} X$ iff $\mathcal{I}(X)([t_1, t_2]) = tt$
2. $\mathcal{M}, [t_1, t_2] \models_{HDC} R^n(\theta_1, \ldots, \theta_n)$
   iff $\mathcal{I}(R^n)(\mathcal{I}(\theta_1)([t_1, t_2]), \ldots, \mathcal{I}(\theta_n)([t_1, t_2])) = tt$
3. $\mathcal{M}, [t_1, t_2] \models_{HDC} \neg\phi$ iff $\mathcal{M}, [t_1, t_2] \not\models_{HDC} \phi$
4. $\mathcal{M}, [t_1, t_2] \models_{HDC} \phi \vee \psi$ iff $\mathcal{M}, [t_1, t_2] \models_{HDC} \phi$ or $\mathcal{M}, [t_1, t_2] \models_{HDC} \psi$
5. $\mathcal{M}, [t_1, t_2] \models_{HDC} \phi; \psi$
   iff $\mathcal{M}, [t_1, t] \models_{HDC} \phi$ and $\mathcal{M}, [t, t_2] \models_{HDC} \psi$ for some $t \in [t_1, t_2]$
6. $\mathcal{M}, [t_1, t_2] \models_{HDC} \exists z.\phi$ iff $< \mathcal{F}, \mathcal{I}' >, [t_1, t_2] \models_{HDC} \phi$
   for some interpretation $\mathcal{I}'$ which is z-equivalent to $\mathcal{I}$ where z is from $Var \cup PVar$.

Satisfaction and validity can be defined as in classical logic, see [3].

4.2 Completeness of HDC on Abstract Domains

[9] has shown that finite variability cannot be axiomatised by finite rules on abstract domains, so DCR1 and DCR2 are not efficient enough to axiomatise the finite variability of program variables on abstract domains. In order to axiomatise the finite variability, we will use a ω -rule [16] instead of DCR1 and DCR2.

Let $\Omega_{hdc} \;\widehat{=}\; \{\exists x.(\llbracket V = x \rrbracket \vee \lceil\rceil) \;\;|\;\; V \in PVar\}$. Let Φ be a formula of HDC. We define a sequence of formulae $\{\Phi^k\}_{k<\omega}$ as follows:

$$\Phi^0 \;\widehat{=}\; \ell = 0, \qquad\qquad \Phi^{k+1} \;\widehat{=}\; (\Phi^k;\Box\Phi)$$

Thus, we can formulate the ω -rule as

$$\mathrm{IR}^{\Phi} \quad \frac{H(\Phi^0/X) \quad \forall k<\omega . H(\Phi^k/X) \Rightarrow H(\Phi^{k+1}/X)}{H(\mathbf{true}/X)} \quad \text{for } \Phi \in \Omega_{hdc}$$

Definition 8 *Let Γ be a set of formulae of HDC. If $\Gamma \not\vdash_{\mathrm{HDC}}$* **false**, *then Γ is* consistent, *otherwise* inconsistent.

We have the following result about the proof system after replacing DCR1 and DCR2 with the ω -rule

Theorem 8 (Soundness and Completeness) *Let Γ be a set of formulae of HDC. Then Γ is consistent if only if Γ is satisfiable.*

Proof: It is easy to prove the soundness, and we will omit it. The proof for the completeness is very complicated, so we only give a sketch of its proof, details can be seen in [8].

A naive way to reduce the second order logic to the first order one is to introduce for the class of n-ary predicates, $H^n(x_1, ..., x_n)$, a new $(n+1)$-ary predicate, $E^{n+1}(z, x_1, ..., x_n)$, which has an additional argument z, and enumerates all $H^n(x_1, ..., x_n)$. Thus,

$$\exists H^n.\phi$$

could be reduced to

$$\exists z.\phi[E^{n+1}(z, x_1, ..., x_n)/H^n(x_1, ..., x_n)]$$

Therefore the second order logic could be reduced to a first order one. Detail discussion about this encoding can be seen in [19]. However, in order to define the $(n+1)$-ary predicate E^{n+1}, we must have the following postulates, where we assume $(n = 1)$ and drop the indices of n and $(n+1)$ for simplicity. Firstly,

$$\exists z.E(z, x_1) \qquad \text{and} \qquad \exists z.\neg E(z, x_1)$$

postulate that, for a singleton domain, E enumerates all H. Furthermore, together with the above two formulae, the formula

$$\exists z.\ (x_1 \neq x_2) \Rightarrow (E(z, x_1) \Leftrightarrow E(z_1, x_1) \wedge E(z, x_2) \Leftrightarrow E(z_2, x_2))$$

postulates that E enumerates all H over any finite domain. Unfortunately, with this approach, we can never define E to enumerate all H over an infinite domain.

However, by the finite variability of program variables, given an interval, any program variables V can be generated by finite combination of subintervals of the given one, over each of which V is constantly. Hence, it is possible to construct a 1-ary flexible function, $g(y)$, to enumerate all program variables by the postulates including

$$\lceil\,\rceil \;\vee\; \exists y.\lceil g(y) = c\rceil \text{ for any constant } c$$

and

$$\lceil\,\rceil \;\vee\; \exists y.\ \lceil g(y) \Leftrightarrow g(y_1)\rceil; \lceil g(y) \Leftrightarrow g(y_2)\rceil$$

In this way, $\exists V.\phi$ can be reduced to $\exists y_V.\phi'$, where ϕ' is ϕ's correspondence in a complete first-order two-sorted interval temporal logic (IL_2). A complete proof system for HDC can be established based on the completeness of IL_2 . This idea is hinted by Dr. Guelev D P [2].

In order to prove completeness of HDC, we will establish IL_2 , a first-order two-sorted interval temporal logic firstly, in which global variables and functions are divided into two sorts. Completeness of IL_2 can be proved using the method provided in [9, 16]. Then we encode HDC into IL_2 . Because we can show that the consistency of a set of formulae Γ in HDC w.r.t. the proof system of HDC implies the consistency of $\Gamma' \cup Axiom_{hdc}$ w.r.t. IL_2 , where Γ' and $Axiom_{hdc}$ stand for the correspondences of Γ and the set of all instances of axioms of HDC in IL_2 respectively. We can get a model $<< T, \leq>, < D+, 0 >, D_1, m, \mathcal{J} >$ which satisfies $\Gamma' \cup Axiom_{hdc}$ by the completeness of IL_2 . According to the model $<< T, \leq>, < D+, 0 >, D_1, m, \mathcal{J} >$, we can construct a model $<< T, \leq>, < D+, 0 >, m, \mathcal{I} >$ for HDC which satisfies Γ. Thus, the completeness of HDC can be proved. □

5 Discussions

In this paper, divergent program behaviour is deliberately avoided. Otherwise, we have to introduce into DC fixed point operators. [20] proposed such operators, and [4] suggests infinite disjunction and conjunction. However, logic foundation for those operators can be further investigated, while they are applied practically.

In the literature of DC, there are two completeness results. One is on abstract domains (see [16]). Unfortunately it requires ω-rule. The other is on real domain (see [15]). It assumes that if $\mathcal{F}$ is a valid formula of ITL, in which only the temporal variables $v_1, \ldots, v_n$ occur, then $\mathcal{D}$ is an axiom of DC, where $\mathcal{D}$ is obtained by replacing each temporal variable v_i with $\int P_i$ where P_i is a state variable. Hence it is a relative completeness. Up to now, no one find a relation between these two completeness results.

If we give another relative completeness of HDC, i.e. if $\models_{HDC} \phi$, then $\Gamma_R \vdash_{HDC} \phi$, where Γ_R stands for all valid formulae of the reals, then we can show that if ITL is complete on real domain w.r.t. the assumption that all valid formulae of the reals are provable, then the completeness of HDC on real domain under the same assumption can be proved with the technique developed in this paper. This conclusion can be applied to other variants of DC too. But how to prove the relative completeness of ITL on real domain is still an open problem.

Acknowledgements: The author sincerely gives his special thanks to Prof. Zhou Chaochen, Dr. Guluev D.P., Prof. He Jifeng and Dr. Xu Qiwen for lots of helps from them during doing this work.

[2]Guelev D P. Quantification over State in Duration Calculus Manuscript, August, 1998.

References

[1] Zhou Chaochen, Hoare C A R, and Ravn A P. A calculus of durations. Information Processing Letters, 1991,40(5):269–276.

[2] Halpern J, Moskowski B, and Manna Z. A hardware semantics based on temporal intervals. In ICALP'83, Springer-Verlag, 1983, LNCS 154, 278–291.

[3] Zhan Naijun. A Higher-Order Duration Calculus and Its Applications . Ph.D Thesis, Institute of Software, the Chinese Academy of Sciences, 2000.

[4] He Jifeng and Xu Qiwen. Advanced features of duration calculus and their applications. UNU/IIST Report No.171, UNU/IIST, P.O. Box 3058, Macau, August, 1999.

[5] Manna Z and Pnueli A. Models of reactitivity. Acta Informatica,Spring-Verlog, 1993, 30(7):609-678.

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

[7] Zhou Chaochen, Guelev D P and Zhan Naijun. A higher-order duration calculus. UNU/IIST Report No. 167, UNU/IIST, P.O. Box 3058, Macau, July, 1999.

[8] Zhan Naijun. Completeness of higher-order duration calculus. UNU/IIST Report No.175, UNU/IIST, P.O. Box 3058, Macau, August, 1999.

[9] Dutertre B. On first order interval temporal logic. Report no. CSD-TR-94-3, Department of Computer Science, Royal Holloway, University of London, Eghan, Surrey TW20 0EX, England, 1995.

[10] Zhou Chaochen, Ravn A P and Hansen M R. An extended duration calculus for hybrid systems. In Grossman R L, Nerode A, Ravn A P, et al, editors, Hybrid Systems, Springer-Verlag, 1993, LNCS 736, 36–59.

[11] Zhou Chaochen and Li Xiaoshan. A mean value calculus of durations. In A Classical Mind: Essays in Honour of C.A.R. Hoare, Prentice Hall, 1994, 431–451.

[12] Liu Zhiming, Ravn A P, Sørensen E V, et al. A probabilistic duration calculus. In Dependable Computing and Fault-Tolerant Systems Vol. 7: Responsive Computer Systems, Springer-Verlag, Wien, New York, 1993, 30–52.

[13] Abadi M. The power of temporal proofs. *Theoretical Computer Science*, 1989, 65: 35-83, *Corrigendum in TCS 70 (1990), page 275*

[14] Garson J W. Quantification in modal logic. In Handbook of Philosophical Logic, Gabbay D and Guenther F (Eds), Reidel, 1984, (II):249-307.

[15] Hansen M R and Zhou Chaochen. Semantics and completeness of duration calculus. In Real-Time: Theory in Practice, Springer-Verlag, 1992,LNCS 600, 209–225.

[16] Guelev. D P. A calculus of durations on abstract domains: completeness and extensions. UNU/IIST Report No. 139, UNU/IIST, P.O. Box 3058, Macau, May, 1998.

[17] He Jifeng. Provably Correct Systems: Modelling of Communication Languages and Design of Optimized Compiler. McGraw-Hill, 1995

[18] Hansen M R and Zhou Chaochen. Duration calculus: logical foundations. Formal Aspects of Computing,1997, 9:283-330.

[19] Epstein R L. The Semantic Foundations of Logic: Predicate Logic, Oxford University Press, Oxford, UK, 1994.

[20] Pandya P K, Wang Hanpin, and Xu Qiwen. Towards a Theory of Sequential Hybrid Programs. In the Proceedings of IFIP TC2/WG2.2,2.3 International Conference on Programming Concepts and Methods (PROCOMET'98), David Gries and Willem-Paul de Roever (eds), Chapman & Hall, 1998, 366-384.