# Projections: A Technique for Verifying Real-Time Programs in Duration Calculus

Dang Van Hung

The United Nations University
International Institute for Software Technology
P.O.Box 3058, Macau

**Abstract.** We present a technique for handling the true synchrony hypothesis and (infinite) loops in real-time programming. The technique is based on the notion of projections and iterations in Duration Calculus. With this technique, the semantics of real-time programs with shared variables is given in a simple way, and the verification can be done using DC* proof system.

## 1 Introduction

Duration Calculus (DC) was introduced by Zhou, Hoare and Ravn in 1991 ([2]) as a logic to specify the requirements for real-time systems. DC has been proved to be a powerful and simple logic for reasoning about real-time systems, and has been used successfully in many case studies, see e.g. [17, 9] [7, 14]. In our recent work [8], we extended DC with iterations and studied a subclass of the extended calculus formulas that can express the regular behaviour of real-time programs in a very natural and intuitive way which can be considered as a design language. Our aim in this paper is to try to use the formulas in this class to give semantic for real-time programs with shared variables.

Verification and analysis of parallel programs with shared variables have attracted a great deal of attention for long time since Owicki and Gries's work in 1976 ([11]). Verification and analysis of real-time parallel programs are even more difficult. Toward this aim, many attempts have been done to use logics for formalising the behaviour of real-time programs. The challenges are as follows:

- How to handle the *true synchrony hypothesis* which says that computation and communication do not take time, only the waiting for external synchronisation or explicit delay statements take time. This hypothesis simplifies the behaviour of real-time systems when time is discrete. When time is continuous, ones have to deal with chopping points.
- How to handle infinite behaviours and finite behaviours in the same logical formalism. Most of the logics cannot express both kinds of behaviours.
- How to handle the loops in the programs that the number of iterations cannot be known before hand.

Among these attempts, some work have used DC as logical framework for formalising (see, e.g. [3, 15, 12, 13]). In order to overcome the above challenges, they have to introduce a special time structure called weakly monotonic time, to extend DC with infinite intervals over weakly monotonic time, and to introduce the recursion in their extended DC. Because of these extensions, the calculus becomes complicated and perhaps losses the completeness. Besides, they have to deal with a special and strange situation in which time stops. This makes their semantics complicated and not easy to read.

Our approach to give the semantics to parallel real-time programs with share variables in this paper is as follows. To handle the loops in the programs, we use iteration in DC$^*$. This notion is not as powerful as recursion, but is much simpler, and is good enough for our purpose. Note that DC$^*$ with simple occurrences of $^*$ is still complete (see [8]). To handle the true synchrony hypothesis, we use the projection over state variables. The idea is that it should says more precisely that the computation and communication do take time, but the time spent in the computation and communication is so small that it is negligible. We introduce a special state variable $V$. $V$ is present at time $t$ iff computation or communication is not being taken at $t$. So the time that $V$ is absent is for computation or communication which is negligible. Projection of the behaviour of the programs on $V$, i.e. removal of the part of time at which $V$ is false will give the behaviour at the real-time. Therefore, the only thing we have to do to handle the true synchrony hypothesis is to replace the metric of time by $\int V$, the duration of $V$ in DC. To handle the infinite behaviour, we do not need any extension of DC to include the neighbourhood or *inf* modality to specify infinite intervals (as in the papers [1, 3]). We handle this matter in the definition of program semantic. That is, we define semantic of a program as a pair of formulas. The first formula specifies the finite behaviours of the program in the interval from the time it starts to the time it terminates. The second formula specifies the infinite behaviours of the program in the all intervals from the time it starts to the time that is big enough. It turns out that in this way, we do not need any extension of DC other than iteration, but we can give a simple and intuitive semantic to the real-time programs with share variables.

The paper is organised as follows. The next section gives a brief of DC$^*$ (DC with iteration) and simple DC$^*$. In Section 3, we define the projection in DC$^*$ and give semantic for real-time programs with share variables in simple DC$^*$. Section 4 presents our verification techniques, and the last section is the conclusion of the paper.

## 2 DC*

Duration Calculus with iteration (DC$^*$) is a conservative extension of Duration Calculus. The readers are referred to [8] for the details of the proof system and the results on completeness and decidability of the calculus. In this section, we give the syntax and semantic of the calculus language.

### 2.1 Language

A language for DC* is built starting from the following sets of *symbols*: a set of *constant symbols* $\{a, b, c, \ldots\}$, a set of *individual variables* $\{x, y, z, \ldots\}$, a set of *state variables* $\{P, Q, \ldots\}$, a set of *function symbols* $\{f, g, \ldots\}$, a set of *relation symbols* $\{R, U, \ldots\}$, These sets are required to be pairwise disjoint and disjoint with the set $\{\mathbf{0}, \bot, \neg, \vee, ^\frown, ^*, \exists, \int, (,)\}$. Besides, 0 should be one of the constant symbols; + should be a binary function symbol; = and $\leq$ should be binary relation symbols.

Given the sets of symbols, a DC* language definition is essentially that of the sets of *state expressions* $S$, *terms* $t$ and *formulas* $\varphi$ of the language. These sets can be defined by the following BNFs:

$$
\begin{array}{l}
S \mathrel{\widehat{=}} \mathbf{0} \mid P \mid \neg S \mid S \vee S \\
t \mathrel{\widehat{=}} c \mid x \mid \int S \mid f(t, \ldots, t) \\
\varphi \mathrel{\widehat{=}} R(t, \ldots, t) \mid \neg\varphi \mid (\varphi \vee \varphi) \mid (\varphi^\frown\varphi) \mid (\varphi^*) \mid \exists x\varphi
\end{array}
$$

### 2.2 Semantics

In this section, we denote by $\mathbf{I}$ the set of the bounded intervals over the set of real numbers $\mathbf{R}$, $\{[\tau_1, \tau_2] \mid \tau_1, \tau_2 \in \mathbf{R}, \tau_1 \leq \tau_2\}$.

Given a DC* language $\mathcal{L}$, a model for $\mathcal{L}$ is an *interpretation* $\mathcal{I}$ of the symbols of $\mathcal{L}$ that satisfies the following conditions: $\mathcal{I}(c), \mathcal{I}(x) \in \mathbf{R}$ for constant symbols $c$ and individual variables $x$; $\mathcal{I}(f) : \mathbf{R}^n \to \mathbf{R}$ for $n$-place function symbols $f$, and $\mathcal{I}(R) : \mathbf{R}^n \to \{0, 1\}$ for $n$-place relation symbols $R$; $\mathcal{I}(P) : \mathbf{R} \to \{0, 1\}$ for state variable $P$ Besides, $\mathcal{I}(0) = 0$

The following condition, known as *finite variability of state*, is imposed on interpretations:
*For every* $[\tau_1, \tau_2] \in \mathbf{I}$ *such that* $\tau_1 < \tau_2$*, and every state variable* $S$ *there exist* $\tau'_1, \ldots, \tau'_n \in \mathbf{R}$ *such that* $\tau_1 = \tau'_1 < \ldots < \tau'_n = \tau_2$ *and* $\mathcal{I}(S)$ *is constant on the intervals* $(\tau'_i, \tau'_{i+1})$*,* $i = 1, \ldots, n-1$.

**Definition 1.** *Given a DC interpretation* $\mathcal{I}$ *for the DC* language* $\mathcal{L}$*, the meaning of state expressions* $S$ *in* $\mathcal{L}$ *under* $\mathcal{I}$*,* $S_\mathcal{I} : \boldsymbol{R} \to \{0, 1\}$*, is defined inductively as follows: for all* $\tau \in \boldsymbol{R}$

$$
\begin{array}{ll}
\mathbf{0}_\mathcal{I}(\tau) \mathrel{\widehat{=}} 0 & \\
P_\mathcal{I}(\tau) \mathrel{\widehat{=}} \mathcal{I}(P)(\tau) & \textit{for state variables } P \\
(\neg S)_\mathcal{I}(\tau) \mathrel{\widehat{=}} 1 - S_\mathcal{I}(\tau) & \\
(S_1 \vee S_2)_\mathcal{I}(\tau) \mathrel{\widehat{=}} \max((S_1)_\mathcal{I}(\tau), (S_2)_\mathcal{I}(\tau)) &
\end{array}
$$

*Given an interval* $[\tau_1, \tau_2] \in \mathbf{I}$*, the meaning of a term* $t$ *in* $\mathcal{L}$ *under* $\mathcal{I}$ *is a number* $\mathcal{I}^{\tau_2}_{\tau_1}(t) \in \mathbf{R}$ *defined inductively as follows:*

$$
\begin{array}{ll}
\mathcal{I}^{\tau_2}_{\tau_1}(c) \mathrel{\widehat{=}} \mathcal{I}(c) & \textit{for constant symbols } c, \\
\mathcal{I}^{\tau_2}_{\tau_1}(x) \mathrel{\widehat{=}} \mathcal{I}(x) & \textit{for individual variables } x, \\
\mathcal{I}^{\tau_2}_{\tau_1}(\textstyle\int S) \mathrel{\widehat{=}} \int_{\tau_1}^{\tau_2} S_\mathcal{I}(\tau) d\tau & \textit{for state expressions } S, \\
\mathcal{I}^{\tau_2}_{\tau_1}(f(t_1, \ldots, t_n)) \mathrel{\widehat{=}} \mathcal{I}(f)(\mathcal{I}^{\tau_2}_{\tau_1}(t_1), \ldots, \mathcal{I}^{\tau_2}_{\tau_1}(t_n)) & \textit{for } n\textit{-place function} \\
 & \textit{symbols } f.
\end{array}
$$

The definitions given so far are relevant to the semantics of DC in general. The extension to the semantics that comes with DC$^*$ appears in the definition of the $\models$ relation below. Let us recall the following tradition relation on interpretations.

**Definition 2.** *Let $\mathcal{I}$, $\mathcal{J}$ be interpretations of the symbols of the same DC$^*$ language $\mathcal{L}$. Let $x$ be a symbol in $\mathcal{L}$. The interpretation $\mathcal{I}$ $x$-*agrees *with the interpretation $\mathcal{J}$ iff $\mathcal{I}(s) = \mathcal{J}(s)$ for all symbols $s$ in $\mathcal{L}$, but possibly $x$.*

**Definition 3.** *Given a DC$^*$ language $\mathcal{L}$, and an interpretation $\mathcal{I}$ of the symbols of $\mathcal{L}$. The relation $\mathcal{I}, [\tau_1, \tau_2] \models \varphi$ for $[\tau_1, \tau_2] \in \mathbf{I}$ and formulas $\varphi$ in $\mathcal{L}$ is defined by induction on the construction of $\varphi$ as follows:*

| | |
|---|---|
| $\mathcal{I}, [\tau_1, \tau_2] \not\models \bot$ | |
| $\mathcal{I}, [\tau_1, \tau_2] \models R(\tau_1, \ldots, \tau_n)$ | *iff* $\mathcal{I}(R)(\mathcal{I}^{\tau_2}_{\tau_1}(t_1), \ldots, \mathcal{I}^{\tau_2}_{\tau_1}(t_n)) = 1$ |
| $\mathcal{I}, [\tau_1, \tau_2] \models \neg\varphi$ | *iff* $\mathcal{I}, [\tau_1, \tau_2] \not\models \varphi$ |
| $\mathcal{I}, [\tau_1, \tau_2] \models (\varphi \vee \psi)$ | *iff either* $\mathcal{I}, [\tau_1, \tau_2] \models \varphi$ *or* $\mathcal{I}, [\tau_1, \tau_2] \models \psi$ |
| $\mathcal{I}, [\tau_1, \tau_2] \models (\varphi^\frown\psi)$ | *iff* $\mathcal{I}, [\tau_1, \tau] \models \varphi$ *and* $\mathcal{I}, [\tau, \tau_2] \models \psi$ *for some* $\tau \in [\tau_1, \tau_2]$ |
| $\mathcal{I}, [\tau_1, \tau_2] \models (\varphi^*)$ | *iff either* $\tau_1 = \tau_2$, *or there exist* $\tau'_1, \ldots, \tau'_n \in \mathbf{R}$ *such that* $\tau_1 = \tau'_1 < \ldots < \tau'_n = \tau_2$ *and* $\mathcal{I}, [\tau'_i, \tau'_{i+1}] \models \varphi$ *for* $i = 1, \ldots, n-1$ |
| $\mathcal{I}, [\tau_1, \tau_2] \models \exists x\varphi$ | *iff* $\mathcal{J}, [\tau_1, \tau_2] \models \varphi$ *for some* $\mathcal{J}$ *that* $x$*-agrees with* $\mathcal{I}$ |

Note that only the modelling relation $\models$, and not the interpretations $\mathcal{I}$, makes the difference between DC$^*$ and DC. Besides, the clauses that define the interpretation of constructs other than $^*$ in DC$^*$ are the same as in DC. This entails that DC$^*$ is a *conservative* extension of DC.

The language definition in Section 2.1 introduces a minimal set of DC$^*$ syntactic elements just in order to enable the concise definition of DC$^*$ semantics in Section 2.2. We use the customary *infix* notation for terms with $+$, and formulas with $\leq$ and $=$ occurring in them. We introduce the constant $\top$, the boolean connectives $\wedge$, $\Rightarrow$ and $\Leftrightarrow$, the relation symbols $\neq$, $\geq$, $<$ and $>$, and the $\forall$ quantifier as abbreviations in the usual way. We assume that boolean connectives bind more tightly than $^\frown$. Since $^\frown$ is associative, we omit parentheses in formulas that contain consecutive occurrences of $^\frown$. Besides, we use the following abbreviations, that are generally accepted in Duration Calculus:

$$
\begin{array}{ll}
\mathbf{1} \mathrel{\widehat{=}} \neg\mathbf{0} & \\
\lceil S \rceil \mathrel{\widehat{=}} \int S = \int \mathbf{1} \wedge \int \mathbf{1} \neq 0 & \\
\Diamond\varphi \mathrel{\widehat{=}} \top^\frown\varphi^\frown\top & \\
\Box\varphi \mathrel{\widehat{=}} \neg\Diamond\neg\varphi & \\
(\varphi^+) \mathrel{\widehat{=}} \varphi^\frown(\varphi^*) & \\
\varphi^0 \mathrel{\widehat{=}} \int \mathbf{1} = 0 & \\
\varphi^k \mathrel{\widehat{=}} \underbrace{\varphi^\frown \ldots ^\frown\varphi}_{k \text{ times}} & \text{for } k > 0
\end{array}
$$

The proof system for DC$^*$ consists of a complete Hilbert-style proof system for first order logic (cf. e.g. [16]), axioms and rules for interval logic (cf. e.g. [5]), Duration Calculus axioms and rules ([6]) and axioms about iteration ([8]). Since the proof system for Interval Logic and Duration Calculus have become well-known, we list here axioms about iteration only.

**Axioms about iteration**

$(DC_1^*)$ $\int \mathbf{1} = 0 \Rightarrow \varphi^*$
$(DC_2^*)$ $(\varphi^* \frown \varphi) \Rightarrow \varphi^*$
$(DC_3^*)$ $(\varphi^* \wedge \psi \frown \top) \Rightarrow (\psi \wedge \int \mathbf{1} = 0 \frown \top) \vee (((\varphi^* \wedge \neg\psi \frown \varphi) \wedge \psi) \frown \top)$.

This proof system does not make DC$^*$ complete. The completeness is achieved when we restrict ourselves to the case that the formulas under $^*$ are simple. We define simple DC$^*$ formulas as follows.

**Definition 4.** Simple *DC* formulas are defined by the following BNF:*

$$\varphi \mathrel{\widehat{=}} [\![S]\!] \mid a \leq \int \mathbf{1} \mid \int \mathbf{1} \leq a \mid (\varphi \vee \varphi) \mid (\varphi \wedge \varphi) \mid (\varphi \frown \varphi) \mid \varphi^*$$

The class of simple formulas forms the design language, and can be translated to a real-time program as shown in the section 4. For simplifying the presentation in the next section, we introduce the notion $PREF(D)$ for any simple formula $D$. Intuitively, $PREF(D)$ is a simple formula that holds for all prefixes of an interval that validates $D$.

**Definition 5.** *Given a simple DC* formula $\varphi$, we define a simple DC* formula $PREF(\varphi)$ as follows.*

1. $PREF([\![S]\!]) \mathrel{\widehat{=}} [\![S]\!]^*$
2. $PREF(a \leq \int \mathbf{1}) \mathrel{\widehat{=}} \int \mathbf{1} \geq 0$
3. $PREF(\int \mathbf{1} \leq a) \mathrel{\widehat{=}} \int \mathbf{1} \leq a$
4. $PREF(\varphi \vee \varphi') \mathrel{\widehat{=}} PREF(\varphi) \vee PREF(\varphi')$
5. $PREF(\varphi \wedge \varphi') \mathrel{\widehat{=}} PREF(\varphi) \wedge PREF(\varphi')$
6. $PREF(\varphi \frown \varphi') \mathrel{\widehat{=}} PREF(\varphi) \vee (\varphi \frown PREF(\varphi'))$
7. $PREF(\varphi^*) \mathrel{\widehat{=}} \varphi^* \frown PREF(\varphi)$

It follows immediately from the definition that

**Proposition 1.** $\varphi \Rightarrow \neg(\neg PREF(\varphi) \frown \top)$.

## 3 Projection over a state and semantic of real-time programs

In fact, with the state durations, Duration Calculus has introduced a very general projection operator over time. Taking the duration $\int P$ of a state expression $P$ over an interval, we ignore the time at which $P$ is absent. So, $\int P$ can be considered as a sort of projection of the interval over the parts that $P$ is present. Following Moszkowski's approach [10], we can define a projection of a formula (a relation on DC formulas and state expressions) as follows.

**Definition 6.** *Let $D$ be a DC* formula (to avoid confusion, we assume that no abbreviation is used in the formula), $H$ be a state expression. Let $D/H$ be the DC formula obtained from $D$ by replacing each duration $\int E$ by $\int(E) \wedge H$, where $E$ is a state expression. $D/H$ is called projection of $D$ over $H$.*

Example: $\lceil H \rceil / H \equiv \int H > 0$, $\lceil \neg H \rceil / H \equiv \bot$, $\lceil \neg H \rceil^* / H \equiv \int H = 0$ (Recall that $\lceil H \rceil \mathrel{\widehat{=}} \int H = \int \mathbf{1} \wedge \int \mathbf{1} > 0$).

So, the projection maps a DC formula to a DC formula syntactically. Semantically, only the part of time intervals at which $H$ is true plays role in the evaluation of the formula $D/H$. This property is shown by the following theorem. Given an interpretation $\mathcal{I}$ such that $\lim_{t\to\infty} \int_0^t H_{\mathcal{I}}(t)dt = \infty$, which means that $\delta(t) \mathrel{\widehat{=}} \int_0^t H_{\mathcal{I}}(t)dt$ (see Fig. 1) is a non-decreasing function from $\mathbf{R}$ onto $\mathbf{R}$ (recall that for a state variable $P$, $P_{\mathcal{I}}(t)$ is the interpretation of $P$ under $\mathcal{I}$, which is a $\{0,1\}$-valued function of time). Let us define an interpretation $\mathcal{I}_H$ by $P_{\mathcal{I}_H}(\delta(t)) \mathrel{\widehat{=}} \min\{P_{\mathcal{I}}(t') \mid \delta(t) = \delta(t')\}$ for all state variables $P$. It can be seen that if $P_{\mathcal{I}}(t)$ is finitely variable then so is $P_{\mathcal{I}_H}(t)$.

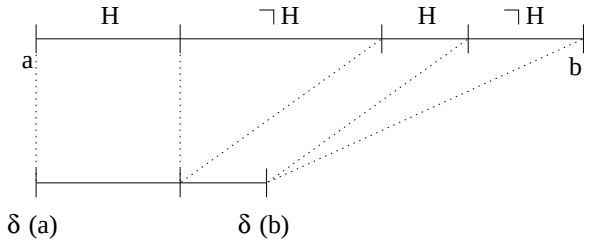


**Fig. 1.** Projection on a state

**Theorem 1.** *For any interpretation $\mathcal{I}$, for any interval $[a,b]$ it holds that $\mathcal{I}, [a,b] \models D/H$ if and only if $\mathcal{I}_H, [\delta(a), \delta(b)] \models D \wedge \lceil H \rceil^*$.*

*Proof.* The proof is based on induction on the structure of the formula $D$. Note that for any state expression $P$ it holds that $\int_a^b (P \wedge H)_{\mathcal{I}}(t)dt = \int_{\delta(a)}^{\delta(b)} P_{\mathcal{I}_H}(t)dt$.

Let $D$ be a formula of the form $R(r_1, \ldots, r_m)$, where $r_1, \ldots, r_m$ are terms, $R$ is a relation. It follows from the definition of semantics that $\mathcal{I}_a^b(r_i[P|(P \wedge H)]) = \mathcal{I}_{H_{\delta(a)}^{\delta(b)}}(r_i)$ for all $i \leq m$. Hence the basic case is verified. Let $D = \neg D'$. Then $\mathcal{I}, [a,b] \models D/H$ iff $\mathcal{I}, [a,b] \models \neg D'/H$ which is the same as $\mathcal{I}, [a,b] \not\models D'/H$. By the inductive hypothesis, this equivalent to $\mathcal{I}_H, [\delta(a), \delta(b)] \not\models D'$. The case is verified. Other cases can be verified trivially in the same way.

Theorem 1 says that the projection is a way to map two formulas and their models accordingly. From this theorem, we can expand a point to an interval for which we can give arbitrary behaviour to the state variables without changing

the truth value of a formula $D$ if we can make sure that $D$ would not refer to the behaviour of the states in that interval. We do so by introducing a new state variable $H$ that is valued to 0 for that interval and to 1 out of it, and then modifying $D$ to $D/H$.

**Theorem 2.** *For any formula $D$, state variables $H, H_1, H_2$ that do not occur in $D$*

1. $\models D$ *if and only if* $\models D/H$
2. $D \equiv D'$ *iff* $(D/H) \equiv (D'/H)$
3. $D/H \wedge \lceil H \rceil \Rightarrow D$
4. $D/H \frown \lceil \neg H \rceil \Rightarrow D/H$
5. $\lceil \neg H \rceil \frown D/H \Rightarrow D/H$
6. $(D' \frown \lceil \neg H \rceil) \wedge D/H \Rightarrow (D' \wedge D/H) \frown \lceil \neg H \rceil$
7. $(\lceil \neg H \rceil \frown D') \wedge D/H \Rightarrow \lceil \neg H \rceil \frown (D' \wedge D/H)$
8. $(D/H_1)/H_2 \equiv D/(H_1 \wedge H_2)$

*Proof.* Straightforward from the definition of DC semantic and Theorem 1.

Note that the projection over a state is just a way to form a class of DC formulas. Given a formula $D$, its projection over a state $H$ is a formula in the class resulting from the substitution $\int(E) \wedge H$ for $\int E$ in $D$. Because the substitution is at the atomic level, it is compositional. Hence, the projection is distributive over all other operators ($\frown, \neg, \wedge, \vee, \exists$). This means that $(D \wedge D')/H = D/H \wedge D'/H$, $(D \frown D')/H = D/H \frown D'/H$, $(D \vee D')/H = (D/H \vee D'/H)$,

**Definition 7.** *A DC* formula $D$ is said to be compositional w.r.t. a state variable $H$ iff $D$ is of the form $D_1/H \wedge D_2/\neg H$ for some $D_1$ and $D_2$.*

The definition can be explained as follows. Suppose that there are two processes in the system. $D_1/H$ and $D_2/\neg H$ are the specification of the first and the second process, respectively. Let $E$ be a DC formula specifying the assumption about environment. Then, the specification of the interleaving composition of the two processes will be $E \wedge D_1/H \wedge D_2/\neg H$. Let $R$ be the requirement of the system. Then, what we have to verify for the system is $E \wedge D_1/H \wedge D_2/\neg H \Rightarrow R$

Now we use simple DC$^*$ and projections to give semantic for real-time programming language with shared variables taking into account the *true synchrony hypothesis*. Because (original) DC formulas cannot specify a point property, our idea is to 'expand' a point to an interval and make the interval to be 'ignored' in the time constraints using projections.

Let a real-time program be of the form $P_1 \parallel \ldots \parallel P_n$, where $P_i$'s are processes. A process $P_i$ is generated by the following grammar:

$$P \mathrel{\widehat{=}} \mathit{skip} \mid x := E(x) \mid \mathit{delay}\ d \mid P; P \mid P \lhd b \rhd P \mid \mathit{while}\ b\ \mathit{do}\ P \mid \mathit{await}\ b$$

where $x$ is a share variable, $b$ is a boolean expression.

*skip* does nothing, *delay* $d$ makes the process wait $d$ time units, $P; P$ is the sequential composition of processes, $P \lhd b \rhd P$ is the conditional branch,

*while b do P* is the loop command, and *await b* make the process waiting until condition $b$ is true.

In order to give semantic to $P$, we introduce a state variable $V$ (visible) to express the non negligible time. For the operations that take no time, we expand the time point at which they happen to an interval of negligible time, i.e. $V$ is absent (0) for the whole interval. For expressing the synchronisation between processes, we assume that each process is associated with state variables $W_i$ and $R_i$. Proposition $R_i$ is true exactly at those time points when actions involving share variables of $P_i$ are taking place (i.e. the $P_i$ needs resources). $W_i$ is true at those time points where the process has terminated.

Each process $P$ will have termination semantic $\mathcal{M}[P]$ and non-termination semantic $\mathcal{M}'[P]$. We use maximal semantic in a sense that the termination semantic will be a DC formula expressing the behaviour in the interval from the beginning to the end of the execution, while the nontermination semantic will be a DC formula expressing the prefixes of the infinite behaviour in all intervals from the beginning to any 'big enough' time point.

*Semantic of skip (in process $P_i$):*

$$\begin{array}{l}\mathcal{M}[skip] \ \widehat{=}\ \lceil\!\lceil \neg R_i \wedge \neg V \rceil\!\rceil \\ \mathcal{M}'[skip] \ \widehat{=}\ false\end{array}$$

*Semantic of $x := E(x)$ (in process $P_i$):*

$$\begin{array}{l}\mathcal{M}[x := E(x)] \ \widehat{=}\ (\lceil\!\lceil \neg R_i \wedge \neg V \rceil\!\rceil^\frown \\ \qquad\qquad ((\exists v)(\lceil\!\lceil R_i \wedge \neg V \rceil\!\rceil \wedge (\lceil\!\lceil x = v \rceil\!\rceil^\frown \lceil\!\lceil x = E(v) \rceil\!\rceil))) \\ \qquad\qquad ^\frown \lceil\!\lceil \neg R_i \wedge \neg V \rceil\!\rceil) \\ \mathcal{M}'[x := E(x)] \ \widehat{=}\ false\end{array}$$

The quantification $(\exists v)$ is indeed the quantification over state variable, not over a global variable. Thus, in fact we have to use the high-order DC$^*$ in this case. For the assignment of the form $x := c$, we can eliminate the use of high-order and define

$$\mathcal{M}[x := c] \ \widehat{=}\ (\lceil\!\lceil \neg R_i \wedge \neg V \rceil\!\rceil^\frown \lceil\!\lceil R_i \wedge \neg V \wedge x = c \rceil\!\rceil^\frown \lceil\!\lceil \neg R_i \wedge \neg V \rceil\!\rceil)$$

The semantic is very intuitive and clear. The process could perform the assignment if it has resources during the execution. The command begins, then action is taking place, and then terminates. The execution is not interfered by any other process (the assignment is atomic). The function of the command is expressed as that if the value at the beginning of the execution is $v$ then on terminating, the value of $x$ will be $E(v)$. The time for executing the command is negligible.

*Semantic of delay d (in process $P_i$):* Delay makes the process stay idle (it could do nothing) for $d$ real time units, and during that time the process should not interfere the others.

$$\mathcal{M}[delay\ d] \ \widehat{=}\ \lceil\neg R_i\rceil \wedge \int V = d$$
$$\mathcal{M}'[delay\ d] \ \widehat{=}\ false$$

It is noted that here, only the non negligible time is counted for the delay.

*Semantic of $P; P'$ (in process $P_i$):*

$$\mathcal{M}[P; P'] \ \widehat{=}\ \mathcal{M}[P]^\frown\mathcal{M}[P']$$
$$\mathcal{M}'[P; P'] \ \widehat{=}\ \mathcal{M}'[P] \vee \mathcal{M}[P]^\frown\mathcal{M}'[P']$$

*Semantic of $P \lhd b \rhd P'$ (in process $P_i$):*

$$\begin{aligned}\mathcal{M}[P \lhd b \rhd P'] \ \widehat{=}\ & \lceil\neg R_i \wedge \neg V\rceil^\frown \\ & ((\lceil b \wedge R_i \wedge \neg V\rceil^\frown\mathcal{M}[P]) \vee (\lceil\neg b \wedge R_i \wedge \neg V\rceil^\frown\mathcal{M}[P'])) \\ \mathcal{M}'[P \lhd b \rhd P'] \ \widehat{=}\ & \lceil\neg R_i \wedge \neg V\rceil^\frown \\ & ((\lceil b \wedge R_i \wedge \neg V\rceil^\frown\mathcal{M}'[P]) \vee (\lceil\neg b \wedge R_i \wedge \neg V\rceil^\frown\mathcal{M}'[P']))\end{aligned}$$

Here we should note that the resources are needed at least during the checking of the condition $b$. After that, the process may still need resources depending on $P$ and $P'$.

*Semantic of while $b$ do $P$ (in process $P_i$):*

$$\begin{aligned}\mathcal{M}[while\ b\ do\ P] \ \widehat{=}\ & \lceil\neg R_i \wedge \neg V\rceil^\frown \\ & ((\lceil b \wedge R_i \wedge \neg V\rceil^\frown\mathcal{M}[P])^*{}^\frown(\lceil\neg b \wedge \neg R_i \wedge \neg V\rceil)) \\ \mathcal{M}'[while\ b\ do\ P] \ \widehat{=}\ & \lceil\neg R_i \wedge \neg V\rceil^\frown( \\ & (((\lceil b \wedge R_i \wedge \neg V\rceil^\frown\mathcal{M}[P])^*)^\frown \\ & \qquad (\lceil b \wedge R_i \wedge \neg V\rceil^\frown\mathcal{M}'[P]))\vee \\ & (\lceil b \wedge R_i \wedge \neg V\rceil^\frown\mathcal{M}[P])^+{}^\frown PREF(\mathcal{M}[P]))\end{aligned}$$

*Semantic of await $b$ (in process $P_i$):* The command *await* $b$ keeps the process waiting until $b$ is true. The condition $b$ may be true when the process starts the command. In this case, the time it waits is negligible. To make the program working in the interleaving style, we enforce that the process should lock the resource for a virtual time period upon the termination of the command.

$$\begin{aligned}\mathcal{M}[await\ b] \ \widehat{=}\ & (\lceil\neg R_i \wedge \neg V\rceil^\frown\lceil b \wedge R_i \wedge \neg V\rceil^\frown\lceil\neg R_i \wedge \neg V\rceil)\vee \\ & (\lceil\neg R_i \wedge \neg b\rceil^\frown\lceil b \wedge R_i \wedge \neg V\rceil^\frown\lceil\neg R_i \wedge \neg V\rceil) \\ \mathcal{M}'[await\ b] \ \widehat{=}\ & \lceil\neg R_i \wedge \neg b\rceil\end{aligned}$$

*Semantic of Programs:* Let us formulate some assumptions of the execution of a program:

Any change of the value of $x$ should be made by an assignment:

$$\mathcal{A}_1 \ \widehat{=}\ (\lceil x = v\rceil^\frown\lceil x = v'\rceil) \wedge v \neq v' \Rightarrow \Diamond\lceil\neg V\rceil$$

Semantic is an interleaving one

$$\mathcal{A}_2 \mathrel{\widehat{=}} \lceil\!\lceil \wedge_{i\neq j} \neg(R_i \wedge R_j) \rceil\!\rceil$$

Program terminates if all processes terminate.

$$\mathcal{A}_3 \mathrel{\widehat{=}} \lceil\!\lceil \wedge_{i\leq n} W_i \rceil\!\rceil \Rightarrow \lceil\!\lceil \neg V \rceil\!\rceil$$

Now the semantic for the program $P_1 \parallel \ldots \parallel P_n$ is given as (we use $\mathcal{S}[P]$ to denote the termination semantic and $\mathcal{S}'[P]$ to denote the nontermination semantic of a program $P$ to distinguish with the semantic of a process)

$$\begin{array}{l} \mathcal{S}[P_1 \parallel \ldots \parallel P_n] \mathrel{\widehat{=}} \mathcal{A}_1 \wedge \mathcal{A}_2 \wedge \mathcal{A}_3 \wedge \wedge_{i\leq n}(\mathcal{M}[P_i]^\frown\lceil\!\lceil \neg R_i \wedge W_i \rceil\!\rceil) \\ \mathcal{S}'[P_1 \parallel \ldots \parallel P_n] \mathrel{\widehat{=}} \mathcal{A}_1 \wedge \mathcal{A}_2 \wedge \mathcal{A}_3 \wedge (\bigvee_{\emptyset\neq J\subseteq\{1,\ldots,n\}} \\ \qquad \wedge_{i\in J}\mathcal{M}'[P_i] \wedge \wedge_{i\in\{1,\ldots,n\}\setminus J}(\mathcal{M}[P_i]^\frown\lceil\!\lceil \neg R_i \wedge W_i \rceil\!\rceil)) \end{array}$$

Advantages of this semantics:

- The semantic is intuitive and simple, using only simple DC formulas, no extension is needed
- Nonterminating behaviour of loop can be modelled precisely
- There is no concept of points, only that of intervals, and it is much easier to handle with intervals than with points.
- It is easy to be converted to the interleaving trace semantics when time is not taken into account.
- Commands in a process are separated by a virtual period in which process in not running
- The semantic is compositional in the sense that it is defined by conjunction of the semantics of its components (and environment assumption)
- As expressive as EDC (Extended Duration Calculus, [4]).

Let us consider the following examples.

*Example 1* Let $P_1 \mathrel{\widehat{=}}$ *delay 1*; $x := x+1$; *delay 1*, $P_2 \mathrel{\widehat{=}}$ *delay 1*; $x := x+2$; *delay 1*. Let $P \mathrel{\widehat{=}} P_1 \parallel P_2$. We expect that $P$ satisfies $D \mathrel{\widehat{=}} (\exists v)((\lceil\!\lceil x = v \rceil\!\rceil \wedge \int \mathbf{1} = 1)^\frown(\lceil\!\lceil x = v+3 \rceil\!\rceil \wedge \int \mathbf{1} = 1))$. Indeed, this can be derived from our semantics.

$$\begin{array}{l} \mathcal{M}[P_1] = (\lceil\!\lceil \neg R_1 \rceil\!\rceil \wedge \int V = 1)^\frown \\ \qquad ((\lceil\!\lceil \neg R_1 \rceil\!\rceil^\frown((\exists v)(\lceil\!\lceil R_1 \rceil\!\rceil \wedge (\lceil\!\lceil x = v \rceil\!\rceil^\frown\lceil\!\lceil x = v+1 \rceil\!\rceil)))^\frown\lceil\!\lceil \neg R_1 \rceil\!\rceil)\wedge \\ \qquad\quad \lceil\!\lceil \neg V \rceil\!\rceil)^\frown \\ \qquad (\lceil\!\lceil \neg R_1 \rceil\!\rceil \wedge \int V = 1) \\ \mathcal{M}[P_2] = (\lceil\!\lceil \neg R_2 \rceil\!\rceil \wedge \int V = 1)^\frown \\ \qquad ((\lceil\!\lceil \neg R_2 \rceil\!\rceil^\frown((\exists v)(\lceil\!\lceil R_2 \rceil\!\rceil \wedge (\lceil\!\lceil x = v \rceil\!\rceil^\frown\lceil\!\lceil x = v+2 \rceil\!\rceil)))^\frown\lceil\!\lceil \neg R_2 \rceil\!\rceil)\wedge \\ \qquad\quad \lceil\!\lceil \neg V \rceil\!\rceil)^\frown \\ \qquad (\lceil\!\lceil \neg R_2 \rceil\!\rceil \wedge \int V = 1) \end{array}$$

Now, it is easy to derive $\mathcal{S}[P]$ taking into account $\mathcal{A}_1$ and $\mathcal{A}_2$.

$$\begin{aligned}\mathcal{M}[P_1] \wedge \mathcal{M}[P_2] \wedge \mathcal{A}_1 \wedge \mathcal{A}_2 \Rightarrow\ & (\exists v)(((\lceil \neg R_1 \wedge \neg R_2 \rceil \wedge \int V = 1 \wedge \lceil x = v \rceil)^\frown \\ & (((((\lceil R_1 \rceil \wedge (\lceil x = v \rceil^\frown \lceil x = v+1 \rceil)))^\frown \\ & \lceil \neg R_1 \vee \neg R_2 \rceil^\frown \\ & (\lceil R_2 \rceil \wedge (\lceil x = v+1 \rceil^\frown \lceil x = v+3 \rceil))) \\ & \qquad \vee \\ & (((\lceil R_1 \rceil \wedge (\lceil x = v \rceil^\frown \lceil x = v+2 \rceil)))^\frown \\ & \lceil \neg R_1 \vee \neg R_2 \rceil^\frown \\ & (\lceil R_2 \rceil \wedge (\lceil x = v+2 \rceil^\frown \lceil x = v+3 \rceil))))) \\ & \wedge \lceil \neg V \rceil)^\frown \\ & (\lceil \neg R_2 \wedge \neg R_1 \rceil \wedge \int V = 1 \wedge \lceil x = v+3 \rceil)) \\ \Rightarrow\ & (\exists v)(((\lceil x = v \rceil \wedge \int V = 1)^\frown \\ & \qquad (\lceil x = v+3 \rceil \wedge \int V = 1))\end{aligned}$$

From Theorem 2, this means that $\mathcal{S}[P] \Rightarrow D/V$.

*Example 2* Let $P \mathrel{\widehat{=}} x := 1; \mathit{delay}\ 1; x := x+1; x := x+1; \mathit{delay}\ 1$. Then,

$$\begin{aligned}\mathcal{M}[x := 1] &= (\lceil \neg R \rceil^\frown (\exists v)(\lceil R \rceil \wedge (\lceil x = v \rceil^\frown \lceil x = 1 \rceil))^\frown \lceil \neg R \rceil) \wedge \\ &\quad \lceil \neg V \rceil \\ \mathcal{M}[\mathit{delay}\ 1] &= \lceil \neg R \rceil \wedge \int V = 1 \\ \mathcal{M}[x := x+1] &= (\lceil \neg R \rceil^\frown (\exists v)(\lceil R \rceil \wedge (\lceil x = v \rceil^\frown \lceil x = v+1 \rceil))^\frown \lceil \neg R \rceil) \wedge \\ &\quad \lceil \neg V \rceil\end{aligned}$$

Therefore (note that $\mathcal{S}[P]$ is different from $\mathcal{M}[P]$),

$$\begin{aligned}\mathcal{S}[P] = & ((\lceil \neg R \rceil^\frown ((\exists v)(\lceil R \rceil \wedge (\lceil x = v \rceil^\frown \lceil x = 1 \rceil))^\frown \lceil \neg R \rceil)) \wedge \\ & \lceil \neg V \rceil)^\frown \\ & (\lceil x = 1 \wedge \neg R \rceil \wedge \int V = 1)^\frown \\ & ((\lceil \neg R \rceil^\frown ((\lceil R \rceil \wedge (\lceil x = 1 \rceil^\frown \lceil x = 2 \rceil))^\frown \lceil \neg R \rceil)) \wedge \lceil \neg V \rceil)^\frown \\ & ((\lceil \neg R \rceil^\frown (\lceil R \rceil^\frown (\lceil x = 2 \rceil^\frown \lceil x = 3 \rceil))\lceil \neg R \rceil) \wedge \lceil \neg V \rceil)^\frown \\ & (\lceil x = 3 \wedge \neg R \rceil \wedge \int V = 1)\end{aligned}$$

From Theorem 2 and $DC^*$ axioms and rules, it follows that

$$\mathcal{S}[P] \Rightarrow \int V = 0^\frown (\lceil \neg R \wedge x = 1 \rceil \wedge \int V = 1)^\frown \int V = 0^\frown (\lceil \neg R \wedge x = 3 \rceil \wedge \int V = 1)$$

Let $P' \mathrel{\widehat{=}} x := 1; \mathit{delay}\ 1; x := x+2; \mathit{delay}\ 1$.

Similarly we have $\mathcal{S}[P'] \Rightarrow \int V = 0^\frown (\lceil \neg R \wedge x = 1 \rceil \wedge \int V = 1)^\frown \int V = 0^\frown (\lceil \neg R \wedge x = 3 \rceil \wedge \int V = 1)$. So, these two programs satisfy the same specification, but they are not equivalent. Of course, $\mathcal{S}'[P] = \mathcal{S}'[P'] = \bot$

## 4 Verification with Projections

In this section, we brief our technique for the verification for the safety requirements of real-time programs in DC. Because we need neither to handle with point intervals nor any special structure of time, the technique is quite simple.

As said earlier, the variables $R_i$, $V$, $W_i$ in the definition of program semantic are introduced in the implementation stage of the development of real-time systems, and they should not occur in the specification stage of the system development. From Theorem 1, we define the satisfaction of a specification $D$ by a program $P$ as follows.

**Definition 8.** *Let $D$ be a DC* formula in which there is no occurrence of the state variable $V$. A program $P$ satisfies $D$ (denoted by $P$ **sat** $D$) iff $\mathcal{S}[P] \Rightarrow D/V$ and $\mathcal{S}'[P] \Rightarrow D/V$*

(The specification should be modified to allow the refinement of time).

Now, the proof of $P$ **sat** $D$ can be done by using the DC proof system. A set of verification rules based on the DC$^*$ proof system might be developed to support the proof. We illustrate our technique via the classical real-time mutual exclusion protocol proposed by Fischer.

Assume that there are two processes. They use a share variable $v$ initialised to 0. The requirement of the protocol is that the two processes are never in their critical section at the same time.

The protocol $P$ is as follows. $P \mathrel{\widehat{=}} P_1 \parallel P_2$, where the process $P_i$ is defined by

$$
\begin{array}{l}
v := 0; \mathit{while\ true\ do} \\
\qquad\qquad \mathit{await}\ v = 0; \\
\qquad\qquad \mathit{delay}\ 1; \\
\qquad\qquad v := i; \\
\qquad\qquad \mathit{delay}\ 2; \\
\qquad\qquad (CS_i; v := 0) \lhd v = i \rhd \mathit{skip};
\end{array}
$$

For simplicity, we overload $CS_i$ to be a state variable specifying the execution of the command $CS_i$. We assign the semantic for the command $CS_i$ as

$$
\begin{array}{l}
\mathcal{M}[CS_i] \mathrel{\widehat{=}} \lceil\neg R_i \wedge \neg V\rceil^\frown\lceil CS_i \wedge V\rceil^\frown\lceil\neg R_i \wedge \neg V\rceil \\
\mathcal{M}'[CS_i] \mathrel{\widehat{=}} \bot
\end{array}
$$

which means that the process should not hold (shared) resources on entering and on outgoing and during the critical section (otherwise, it is trivial). The critical section takes time. The requirement for the protocol is that the two processes should not be in the critical section at the same time for a nonzero period of time. We formalise the requirement as the DC formula

$$R \mathrel{\widehat{=}} \Box\neg(\lceil CS_1\rceil \wedge \lceil CS_2\rceil)$$

First we compute the semantic $\mathcal{M}'[P_i]$ ($\mathcal{M}[P_i] = \bot$).

$$
\begin{array}{rl}
\mathcal{M}'[P_i] \mathrel{\widehat{=}} & ((\lceil\neg R_i \wedge \neg V\rceil^\frown((\exists c)(\lceil R_i \wedge \neg V\rceil \wedge (\lceil v = c\rceil^\frown\lceil v = 0\rceil)))^\frown \\
& \lceil\neg R_i \wedge \neg V\rceil)^\frown \\
& (\lceil\neg R_i \wedge \neg V\rceil^\frown((\lceil R_i \wedge \neg V\rceil^\frown\mathcal{M}[B_i])^*)^\frown \\
& (\lceil R_i \wedge \neg V\rceil^\frown\mathcal{M}'[B_i])))\vee \\
\\
& ((\lceil\neg R_i \wedge \neg V\rceil^\frown((\exists c)(\lceil R_i \wedge \neg V\rceil \wedge (\lceil v = c\rceil^\frown\lceil v = 0\rceil)))^\frown \\
& \lceil\neg R_i \wedge \neg V\rceil)^\frown \\
& (\lceil\neg R_i \wedge \neg V\rceil^\frown(\lceil R_i \wedge \neg V\rceil^\frown\mathcal{M}[B_i])^+ {}^\frown PREF(\mathcal{M}[B_i])))
\end{array}
$$

where $B_i$ is the body of the *while* loop.

$$\mathcal{M}'[B_i] = \lceil\!\lceil \neg R_i \wedge v \neq 0 \rceil\!\rceil$$

$$\begin{aligned}
\mathcal{M}[B_i] \;= & ((\lceil\!\lceil \neg R_i \wedge \neg V \rceil\!\rceil^\frown \lceil\!\lceil R_i \wedge v = 0 \wedge \neg V \rceil\!\rceil^\frown \lceil\!\lceil \neg R_i \wedge \neg V \rceil\!\rceil) \vee \\
& (\lceil\!\lceil \neg R_i \wedge v \neq 0 \rceil\!\rceil^\frown \lceil\!\lceil R_i \wedge v = 0 \wedge \neg V \rceil\!\rceil^\frown \lceil\!\lceil \neg R_i \wedge \neg V \rceil\!\rceil))^\frown \\
& (\lceil\!\lceil \neg R_i \rceil\!\rceil \wedge \textstyle\int V = 1)^\frown \\
& (\lceil\!\lceil \neg R_i \wedge \neg V \rceil\!\rceil^\frown ((\exists c)(\lceil\!\lceil R_i \wedge \neg V \rceil\!\rceil \wedge (\lceil\!\lceil x = c \rceil\!\rceil^\frown \lceil\!\lceil x = i \rceil\!\rceil)))^\frown \lceil\!\lceil \neg R_i \wedge \neg V \rceil\!\rceil)^\frown \\
& (\lceil\!\lceil \neg R_i \rceil\!\rceil \wedge \textstyle\int V = 2)^\frown \\
& \lceil\!\lceil \neg R_i \wedge \neg V \rceil\!\rceil^\frown \\
& \quad (((\lceil\!\lceil v = i \wedge R_i \wedge \neg V \rceil\!\rceil^\frown \lceil\!\lceil \neg R_i \wedge \neg V \rceil\!\rceil^\frown \\
& \quad \lceil\!\lceil R_i \wedge CS_i \wedge V \rceil\!\rceil^\frown \lceil\!\lceil \neg R_i \wedge \neg V \rceil\!\rceil^\frown \\
& \quad ((\exists c)(\lceil\!\lceil R_i \wedge \neg V \rceil\!\rceil \wedge (\lceil\!\lceil x = c \rceil\!\rceil^\frown \lceil\!\lceil x = 0 \rceil\!\rceil)))^\frown \lceil\!\lceil \neg R_i \wedge \neg V \rceil\!\rceil) \vee \\
& \quad (\lceil\!\lceil v \neq i \wedge R_i \wedge \neg V \rceil\!\rceil^\frown \lceil\!\lceil \neg R_i \wedge \neg V \rceil\!\rceil))
\end{aligned}$$

Now, it is not difficult to see that

$$\begin{aligned}
\mathcal{S}'[P_1 \parallel P_2] &= \mathcal{A}_1 \wedge \mathcal{A}_2 \wedge \mathcal{M}'[P_1] \wedge \mathcal{M}'[P_2] \\
&= \mathcal{A}_1 \wedge \mathcal{A}_2 \wedge \\
&\qquad \bigwedge_{i=1}^{2} \begin{pmatrix} (\lceil\!\lceil \neg R_i \wedge \neg V \rceil\!\rceil^\frown ((\exists c)(\lceil\!\lceil R_i \wedge \neg V \rceil\!\rceil \wedge (\lceil\!\lceil v = c \rceil\!\rceil^\frown \lceil\!\lceil v = 0 \rceil\!\rceil)))^\frown \\ \lceil\!\lceil \neg R_i \wedge \neg V \rceil\!\rceil)^\frown \\ (\lceil\!\lceil \neg R_i \wedge \neg V \rceil\!\rceil^\frown (\lceil\!\lceil R_i \wedge \neg V \rceil\!\rceil^\frown \mathcal{M}[B_i])^+ {}^\frown PREF(\mathcal{M}[B_i])) \end{pmatrix}
\end{aligned}$$

We will give here a sketch for the proof of the correctness of the protocol only.

**Lemma 1.** *For* $i = 1, 2$

1. $\mathcal{S}'[P_1 \parallel P_2] \Rightarrow \Box(\lceil\!\lceil v = 0 \rceil\!\rceil^\frown \lceil\!\lceil v \neq i \rceil\!\rceil^\frown \lceil\!\lceil v = i \rceil\!\rceil \Rightarrow \int V \leq 1)$
2. $\mathcal{S}'[P_1 \parallel P_2] \Rightarrow \Box(\lceil\!\lceil v = i \rceil\!\rceil^\frown \neg \lceil\!\lceil v = i \rceil\!\rceil^\frown \lceil\!\lceil v = i \rceil\!\rceil \Rightarrow \int V \geq 3)$

**Lemma 2.** *For* $i = 1, 2$

1. $\mathcal{S}'[P_1 \parallel P_2] \Rightarrow \Box(\lceil\!\lceil v = i \rceil\!\rceil^\frown \int V \leq 2^\frown \lceil\!\lceil v = i \rceil\!\rceil \Rightarrow \lceil\!\lceil v = i \rceil\!\rceil)$
2. $\mathcal{S}'[P_1 \parallel P_2] \Rightarrow \Box(\lceil\!\lceil CS_i \rceil\!\rceil \Rightarrow \lceil\!\lceil v = i \rceil\!\rceil)$

It follows immediately from Lemma 2 that

**Theorem 3.**

$$\mathcal{S}'[P_1 \parallel P_2] \Rightarrow \Box((\lceil\!\lceil CS_1 \Rightarrow V \rceil\!\rceil \wedge \lceil\!\lceil CS_2 \Rightarrow V \rceil\!\rceil) \wedge \neg(\lceil\!\lceil CS_1 \rceil\!\rceil \wedge \lceil\!\lceil CS_2 \rceil\!\rceil))$$

It should be noticed here that according to Theorem 2, $\Box((\lceil\!\lceil CS_1 \Rightarrow V \rceil\!\rceil \wedge \lceil\!\lceil CS_2 \Rightarrow V \rceil\!\rceil) \wedge \neg(\lceil\!\lceil CS_1 \rceil\!\rceil \wedge \lceil\!\lceil CS_2 \rceil\!\rceil)) \Rightarrow \Box\neg(\int CS_1 = \int V \wedge \int CS_2 = \int V \wedge \int V > 0)$, which says that $P_1 \parallel P_2$ **sat** $R$.

## 5 Conclusion

We have presented our technique to overcome the challenges in giving semantics to real-time parallel programs with shared variables. The technique is simple and based on the classical notions of regular expressions and projections. The semantic of real-time parallel programs with shared variables given in the papers is quite natural and flexible in a sense that it is easy to modify according to the implementation policy. Our work in the future is to give a set of rules for reasoning about real-time programs with shared variables based on this technique.

## References


1. Zhou Chaochen and Michael R. Hansen. An Adequate First Interval Order Logic. Research Report 91, UNU/IIST, P.O.Box 3058, Macau, December 1996. Published in *International Symposium, Compositionality – The Significant Difference*, Hans Langmaack and Amir Pnueli and Willem-Paul de Roever (eds), Springer–Verlag, 1998.
2. Zhou Chaochen, C.A.R. Hoare, and Anders P. Ravn. A calculus of durations. *Information Processing Letters*, 40(5):269–276, 1991.
3. Zhou Chaochen, Dang Van Hung, and Li Xiaoshan. A Duration Calculus with Infinite Intervals. Research Report 40, UNU/IIST, P.O.Box 3058, Macau, February 1995. Published in: *Fundamentals of Computation Theory*, Horst Reichel (ed.), pp. 16-41, LNCS 965, Springer-Verlag, 1995.
4. Zhou Chaochen, Anders P. Ravn, and Michael R. Hansen. An Extended Duration Calculus for Real-time Systems. Research Report 9, UNU/IIST, P.O.Box 3058, Macau, January 1993. Published in: *Hybrid Systems*, LNCS 736, 1993.
5. B. Dutertre. On First Order Interval Temporal Logic. Technical Report CSD-TR-94-3, Department of Computer Science, Royal Holloway, University of London, Egham, Surrey TW20 0EX, England, 1995.
6. Michael R. Hansen and Zhou Chaochen. Semantics and completeness of duration calculus. In J.W. de Bakker, C. Huizing, W.P. de Roever, and G. Rozenberg, editors, *Real-Time Systems: Theory in Practice*, Lecture Notes in Computer Science 600, pages 209–226. Spriger-Verlag, 1992.
7. Dang Van Hung. Modelling and Verification of Biphase Mark Protocols in Duration Calculus Using PVS/DC$^-$. Research Report 103, UNU/IIST, P.O.Box 3058, Macau, April 1997. Presented at and published in the Proceedings of the *1998 International Conference on Application of Concurrency to System Design* (CSD'98), 23-26 March 1998, Aizu-wakamatsu, Fukushima, Japan, IEEE Computer Society Press, 1998, pp. 88 - 98.
8. Dang Van Hung and Dimitar P. Guelev. Completeness and Decidability of a Fragment of Duration Calculus with Iteration. Technical Report 163, UNU/IIST, P.O.Box 3058, Macau, April 1999. Presented at International Conference on Mathematical Foundation of Informatics, Hanoi, October 25-28, 1999. Presented at and published in the proceedings of Asian Computing Science Conference (ASIAN'99), Phuket, Thailand, December 10-12, 1999, P.S. Thiagarajan and R. Yap (eds), *Advances in Computing Science*, LNCS 1742, Springer-Verlag, 1999, pp. 139–150.

9. Dang Van Hung and Wang Ji. On The Design of Hybrid Control Systems Using Automata Model. Research Report 35, UNU/IIST, P.O.Box 3058, Macau, November 1994. Published in V. Chandru and V. Vinay (Eds.) *Foundations of Software Technology and Theoretical Computer Science* (FST&TCS16), LNCS 1180, Springer-Verlag, Dec 1996, pp. 156–167.
10. Ben Moszkowski. Compositional Reasoning about Projected and Infinite Intervals. Technical Report EE-0495-M1, Department of Electrical and Electronic Engineering, University of Newcastle upon Tyne, Newcastle NE1 7RU, UK, April 1995.
11. Susan Owicki and David Gries. An axiomatic proof technique for parallel programs. *Acta Informatica*, pages 319–340, 1976.
12. Paritosh K. Pandya and Dang Van Hung. Duration Calculus with Weakly Monotonic Time. Technical Report 122, UNU/IIST, P.O.Box 3058, Macau, September 1997. Presented at and published in the proceedings of *Formal Techniques in Real-Time and Fault-Tolerant Systems* 5th International Symposium, Lyngby, Denmark, September 1998 (FTRTFT'98), Anders P. Ravn and Hans Rischel (Eds.), LNCS 1486, pp. 55–64, Springer-Verlag, 1998.
13. P.K. Pandya, Wang Hanpin, and Xu Qiwen. Towards a Theory of Sequential Hybrid Programs. Technical Report 125, UNU/IIST, P.O.Box 3058, Macau, October 1997. Presented at and published in the proceedings of IFIP TC2/WG2.2,2.3 International Conference on *Programming Concepts and Methods* (PROCOMET'98), 8-12 June 1998, Shelter Island, New York, USA, David Gries and Willem-Paul de Roever (eds), Chapman & Hall, 1998, pp. 366–384.
14. Ekaterina Pavlova and Dang Van Hung. A Formal Specification of the Concurrency Control in Real-Time Databases. Technical Report 152, UNU/IIST, P.O.Box 3058, Macau, January 1999. Published in the proceedings of 6th Asia-Pacific Software Engineering Conference (APSEC'99) Takamatsu, Japan, December 7-10, 1999, IEEE Computer Society Press, pp. 94–101.
15. Xu Qiwen and Mohalik Swarup. Compositional Reasoning Using Assumption - Commitment Paradigm. Technical Report 136, UNU/IIST, P.O.Box 3058, Macau, February 1998. Published in *International Symposium, Compositionality – The Significant Difference*, Hans Langmaack and Amir Pnueli and Willem-Paul de Roever (eds), LNCS 1536, pp. 565–583, Springer–Verlag, 1998.
16. J. Shoenfield. *Mathematical logic*. Addison-Wesley, Reading, Massachusetts, 1967.
17. Zheng Yuhua and Zhou Chaochen. A Formal Proof of a Deadline Driven Scheduler. Research Report 16, UNU/IIST, P.O.Box 3058, Macau, 1. April 1994. Published in: *Formal Techniques in Real-Time and Fault-Tolerant Systems*, LNCS 863, 1994, pp. 756–775.