Under consideration for publication in Formal Aspects of Computing

Advanced Features of Duration Calculus and Their Applications in Sequential Hybrid Programs[1]

He Jifeng [2] and Xu Qiwen[3]

[2]International Institute for Software Technology, United Nations University, Macao SAR, P.R. China

[3]Faculty of Science and Technology, University of Macau, Macao SAR, P.R. China

Keywords: Duration Calculus; Semantics; Verification; Hybrid systems

Abstract. Duration Calculus was introduced as a logic to specify real-time requirements of computing systems. It has been used successfully in a number of case studies. Moreover, many variants were proposed to deal with various features of real time systems, including sequential communicating processes, sequential hybrid systems and imperative programming languages. This paper aims to integrate several variants of Duration Calculus, and to provide a semantic framework for real-time programming languages and sequential hybrid programs.

1. Introduction

Hybrid systems are interactive systems of continuous devices and real-time control programs. The design of a real-time control system is ideally decomposed into a progression of related phases. It starts with an analysis of requirements and properties of the process evolving within its environment. From these are derived formal specifications of the components of the system. A high level program generated in the later phase of the project is usually translated into machine code of the chosen computer. Additional application-specific hardware components may be needed to embed the computer into the system which it controls. Reliability of

[1] A short version of this paper appeared in J. Davis, A.W. Roscoe and J.C.P Woodcock editors, Millennial Perspectives in Computer Science, Proceedings of the 1999 Oxford-Microsoft Symposium in Honour of Professor C.A.R. Hoare.

[2] On leave from East China Normal University, Shanghai, P.R. China.

the delivered system requires that all the conceptual gaps between specification and implementation be closed.

For hybrid systems, a variety of formal methods have been developed, among them Phase Transition System [HCH93, MMP92], Declarative Control [Koh88], the Extended State-Transition Graph [Nic92] and the Hybrid CSP [He93]. However, it remains a difficult task to mix the description of quantitative timing properties with that of discrete changes of sequential systems. An interval temporal logic with discrete time was investigated for presenting the kinds of temporal properties and signal transitions that occur in real-time control programs. The behaviour of hardware devices can often be decomposed into successively smaller intervals of activity [HMM83, Mos85]. Moreover, state transitions of programs can also be characterised by properties relating the initial and final values of variables over interval of times [Hal83, Mos86]. But in the treatment of hybrid systems where the physical world evolve continuously, this approach seems inappropriate. Furthermore, we lose some important algebraic laws of the Guarded Command Language [Dij76] in that framework, such as induction rules for recursions and the combination of assignments

$$(x := e; x := f) \;=\; (x := f[e/x])$$

This paper proposes a conservative extension of the untimed refinement calculus [HoH98, Jon90, Mor94] for hybrid systems, based on the model of the Duration Calculus (DC) [ZHR91, ZRH93]. Our research is inspired by the pioneering work presented in [PaR95, PaD88, PWX98, ZH96]. The main contribution of this paper includes

1. some novel features which play the vital role in formalising stable state of program variables and sequential composition operator,
2. a link between the untimed refinement calculus and our timed one, which preserves the laws of untimed programming,
3. a theory of *Sequential Hybrid Programs* (SHP).

The advantage of a model-oriented calculus is that it describes as directly as possible, using the full expressive power of mathematics, the observable and testable properties of the desired systems. These properties can be specified in independent modules, and can be assembled by simple combinators. Because both design notation and implementation languages are also given model-oriented semantics, correctness can be proved by deduction.

The remainder of the paper is organised as follows. Section 2 gives a brief account of DC. We introduce some advanced features and explore their algebraic properties in Section 3. Section 4 presents a link between the untimed refinement calculus [Jon90, HoH98, Mor94] and the timed one, and shows that the embedding from the former to the latter is a homomorphism. Section 5 presents a theory of Sequential Hybrid Programs. Those programs consist of phase statements and the conventional sequential programming constructs, where the former is used to specify the dynamic behaviours of controlled devices, and the latter describes the activity of control program. We provide a compositional semantics for SHP and study verification methods based on the advanced features of DC introduced in the earlier sections.

2. Preliminaries

In DC, time is continuous and represented by reals

$$Time \ =_{df} \ Real$$

An interval is represented by $[t1, t2]$ where $t1, t2 \in Time$ and $t1 \leq t2$. We use I to stand for the set of intervals, and σ to range over intervals, and $\sigma.b$ and $\sigma.e$ to represent its left and right end points. Adjacent intervals can be merged using the concatenation operator $\hat{}$.

$$\sigma_1 \,\hat{}\, \sigma_2 \ =_{df} \ \sigma_1 \cup \sigma_2 \qquad \text{if} \ \ \sigma_1.e = \sigma_2.b$$

A model $\mathcal{M}$ assigns every symbol a constant of its associated type:

- Global variable x is assigned a real $\mathcal{M}(x)$.
- State variable V is interpreted as a time function $\mathcal{M}(V) : Time \rightarrow Real$.
- Temporal variable v is associated with an interval function $\mathcal{M}(v) : I \rightarrow Real$. A specific temporal variable ℓ denotes the length of interval

  $$\mathcal{M}(\ell)(\sigma) \ =_{df} \ \sigma.e - \sigma.b$$

- $\mathcal{M}$ assigns an n-ary function name f a function $\mathcal{M}(f) : Real^n \rightarrow Real$.
- An n-ary predicate name p is associated with a predicate

  $$\mathcal{M}(p) : Real^n \rightarrow Bool$$

Let h be a variable. Two models $\mathcal{M}$ and $\mathcal{M}0$ are said to be *h-equivalent*, denoted by $\mathcal{M} \equiv_h \mathcal{M}0$, if for all variables v different from h

$$\mathcal{M}(v) \ = \ \mathcal{M}0(v)$$

The terms of the language are defined by induction, and interpreted as real-valued functions over intervals.

- Global and temporal variables are terms.
- If $r_1, \ldots, r_n$ are terms and f is an n-ary function name, then $f(r_1, \ldots, r_n)$ is also a term

  $$\mathcal{M}(f(r_1, \ldots, r_n))(\sigma) \ =_{df} \ \mathcal{M}(f)(\mathcal{M}(r_1)(\sigma), \ldots, \mathcal{M}(r_n)(\sigma))$$

Formulae are interpreted as functions from intervals to the Boolean values *true* and *false*. The set of well-formed formulae is generated by the following rules:

- **true** and **false** are well-formed formulae.

  $$\mathcal{M}(\mathbf{true})(\sigma) \ =_{df} \ true$$

  $$\mathcal{M}(\mathbf{false})(\sigma) \ =_{df} \ false$$

- If $r_1, \ldots, r_n$ are terms, and p is an n-ary predicate name, then $p(r_1, \ldots, r_n)$ is a well-formed formula

  $$\mathcal{M}(p(r_1, \ldots, r_n))(\sigma) \ =_{df} \ \mathcal{M}(p)(\mathcal{M}(r_1)(\sigma), \ldots, \mathcal{M}(r_n(\sigma))$$

- If F and G are well-formed formulae, so are $\neg F$ and $F \wedge G$, $F \,\hat{}\, G$ and $\exists h \bullet F$, where h is a global variable.

  $$\mathcal{M}(F \,\hat{}\, G)(\sigma) \ =_{df} \ \exists \sigma_1, \sigma_2 \bullet \sigma \ = \ (\sigma_1 \,\hat{}\, \sigma_2) \ \wedge \ \mathcal{M}(F)(\sigma_1) \ \wedge \ \mathcal{M}(G)(\sigma_2)$$

$$\mathcal{M}(\exists h \bullet F)(\sigma) \ =_{df} \ \exists \mathcal{M}0 \bullet \ \mathcal{M}0(F)(\sigma) \ \wedge \ (\mathcal{M} \equiv_h \mathcal{M}0)$$

All the usual logical connectives (disjunction, implication, etc.) and quantifiers can be defined

$$\begin{aligned} F \vee G \ &=_{df} \ \neg\,(\neg F \ \wedge \ \neg G) \\ F \Rightarrow G \ &=_{df} \ \neg F \ \vee \ G \\ \forall h \bullet F \ &=_{df} \ \neg(\exists h \bullet \neg F) \end{aligned}$$

The formula $F \lhd H \rhd G$ presents a choice between F and G depending on H.

$$F \lhd H \rhd G \ =_{df} \ (F \wedge H) \vee (G \wedge \neg H)$$

The modal operators $\Diamond$ and $\Box$ can be defined in terms of the chop operator. The formula $\Diamond F$ holds on an interval if F does so on one of its subintervals.

$$\Diamond F \ =_{df} \ \mathbf{true} \,\hat{}\, (F \,\hat{}\, \mathbf{true})$$

The formula $\Box F$ holds if F holds on all its subintervals.

$$\Box F \ =_{df} \ \neg\Diamond(\neg F)$$

3. Advanced Features

This section introduces a set of advanced features, which play the vital role later in modelling sequential hybrid programs.

3.1. Initial and Final Values

Definition 3.1 (Initial and final values)

Let S be a state expression. We introduce two temporal terms $\mathbf{b}.S$ and $\mathbf{e}.S$ defined by

$$\begin{aligned} \mathcal{M}(\mathbf{b}.S)(\sigma) \ &=_{df} \ \mathcal{M}(S)(\sigma.b) \\ \mathcal{M}(\mathbf{e}.S)(\sigma) \ &=_{df} \ \mathcal{M}(S)(\sigma.e) \end{aligned} \qquad \Box$$

Example 3.2

Let V be a state variable. Formula $\mathbf{stable}(V)$ holds on an interval if the value of V is constant throughout that interval

$$\mathbf{stable}(V) \ =_{df} \ \Box(\mathbf{b}.V \ = \ \mathbf{e}.V)$$

$\mathbf{stable}^{-}(V)$ holds on an interval if V remains unchanged except at the end of the interval

$$\mathbf{stable}^{-}(V) \ =_{df} \ \neg((\mathbf{b}.V \neq \mathbf{e}.V) \,\hat{}\, (\ell > 0)) \qquad \Box$$

Theorem 3.3 $\mathbf{b}.V$ and $\mathbf{e}.V$ are subject to the following law:

(V-1) $\quad (F \wedge p(\mathbf{e}.V)) \,\hat{}\, G \ = \ F \,\hat{}\, (p(\mathbf{b}.V) \wedge G)$

(V-2) $\quad (p(\mathbf{b}.V) \wedge F) \,\hat{}\, G \ = \ p(\mathbf{b}.V) \wedge (F \,\hat{}\, G)$

(V-3) $\quad F \frown (G \wedge p(\mathbf{e}.V)) \;=\; (F \frown G) \wedge p(\mathbf{e}.V)$

(V-4) $\quad (\ell = 0) \wedge p(\mathbf{e}.V) \;=\; p(\mathbf{b}.V) \wedge (\ell = 0)$

3.2. Stability

Definition 3.4 (Stability)
Let S be a Boolean state expression. Define

$$\lfloor S \rfloor \;=_{df}\; \neg((\ell > 0) \frown \neg\mathbf{b}.S \frown (\ell > 0))$$

It expresses that S holds everywhere inside the interval. $\lfloor\ \rfloor$ enjoys the following algebraic properties.

Theorem 3.5
(1) $\lfloor true \rfloor \;=\; \mathbf{true}$

(2) $\lfloor false \rfloor \;=\; (\ell = 0)$

(3) $\lfloor S \wedge T \rfloor \;=\; \lfloor S \rfloor \wedge \lfloor T \rfloor$

(4) $\lfloor S \rfloor \;=\; \Box(\lfloor S \rfloor)$

Proof. we only show (4).

$$\begin{array}{lll}
 & \Box(\lfloor S \rfloor) & \{\text{Def of } \Box\} \\
= & \neg(\mathbf{true} \frown \neg(\lfloor S \rfloor) \frown \mathbf{true}) & \{\text{Def of } \lfloor\ \rfloor\} \\
= & \neg(\mathbf{true} \frown ((\ell > 0) \frown \neg\mathbf{b}.S \frown (\ell > 0)) \frown \mathbf{true}) & \{\text{associativity of } \frown\} \\
= & \neg((\ell > 0) \frown \neg\mathbf{b}.S \frown (\ell > 0)) & \{\text{Def of } \lfloor\ \rfloor\} \\
= & \lfloor S \rfloor & \Box
\end{array}$$

Notation
We use the notation $\|S\|$ to abbreviate

$$\mathbf{b}.S \;\wedge\; \lfloor S \rfloor \;\wedge\; \mathbf{e}.S \qquad \Box$$

3.3. Left and Right Neighbourhood Values

A state variable V is *finitary* if it can only change a finite number of times in any finite interval. We define its left neighbourhood value $\overleftarrow{V}$ and right neighbourhood value $\overrightarrow{V}$ by

$$\mathcal{M}(\overleftarrow{V})(\sigma) \;=\; c \quad \text{if} \quad \exists\, \delta > 0 \bullet \mathcal{M}(\lfloor V = c \rfloor)([\sigma.b - \delta,\, \sigma.b]) = \mathit{true}$$

$$\mathcal{M}(\overrightarrow{V})(\sigma) \;=\; c \quad \text{if} \quad \exists\, \delta > 0 \bullet \mathcal{M}(\lfloor V = c \rfloor)([\sigma.e,\, \sigma.e + \delta]) = \mathit{true}$$

For a state expression E, $\overleftarrow{E}$ and $\overrightarrow{E}$ can be defined in the same way. $\Box$

The concepts of neighbourhood value are captured by the following laws:

(nv-1) $(F \wedge p(\overrightarrow{V})) \frown (G \wedge (\ell > 0))$
$= F \frown (G \wedge \exists x \bullet (p(x) \wedge ((\lfloor V = x \rfloor \wedge \ell > 0) \frown \mathbf{true})))$

$(F \wedge (\ell > 0)) \frown (p(\overleftarrow{V}) \wedge G)$
$= F \wedge \exists x \bullet (p(x) \wedge (\mathbf{true} \frown (\lfloor V = x \rfloor \wedge \ell > 0))) \frown G$

(nv-2) $F \frown (G \wedge p(\overrightarrow{V})) = (F \frown G) \wedge p(\overrightarrow{V})$

$(F \wedge p(\overrightarrow{V})) \frown (G \wedge (\ell = 0)) = (F \frown (G \wedge (\ell = 0))) \wedge p(\overrightarrow{V})$

(nv-3) $(F \wedge (\ell = 0)) \frown (p(\overleftarrow{V}) \wedge G) = p(\overleftarrow{V}) \wedge ((F \wedge (\ell = 0)) \frown G)$

$(p(\overleftarrow{V}) \wedge F) \frown G = p(\overleftarrow{V}) \wedge (F \frown G)$

3.4. Hiding State Variable

Definition 3.6 (Hiding a state variable)
Let V be a state variable. Define

$$\mathcal{M}(\exists V \bullet F)(\sigma) =_{df} \exists \mathcal{M}0 \bullet \mathcal{M}0(F)(\sigma) \wedge (\mathcal{M} \equiv_V \mathcal{M}0) \qquad \square$$

The following laws apply:

($\exists$-1) If $(\overrightarrow{V} \notin F)$ and $(\overleftarrow{V} \notin G)$, and $(\mathbf{e}.V \notin F)$ or $(\mathbf{b}.V \notin G)$, then

$$(\exists V \bullet F) \frown (\exists V \bullet G) = \exists V \bullet (F \frown G)$$

($\exists$-2) If none of $\{V, \overleftarrow{V}, \overrightarrow{V}\}$ occurs in both F and G, then

$$\exists V \bullet (F \wedge G) = (\exists V \bullet F) \wedge (\exists V \bullet G)$$

($\exists$-3) If the state variable V is not mentioned in G, then

$$(\exists V \bullet F) \frown G = \exists V \bullet (F \frown G)$$
$$G \frown (\exists V \bullet F) = \exists V \bullet (G \frown F)$$

($\exists$-4) $F = \exists W \bullet (F[W/V] \wedge \lfloor\lfloor W = V \rfloor\rfloor \wedge (\overleftarrow{W} = \overleftarrow{V}) \wedge (\overrightarrow{W} = \overrightarrow{V}))$

Corollary
If $\overrightarrow{V} \notin F$ and $\overleftarrow{V} \notin G$, then

$$F = \exists W \bullet (F[W/V] \wedge \lfloor\lfloor W = V \rfloor\rfloor \wedge (\overleftarrow{W} = \overleftarrow{V}))$$
$$G = \exists W \bullet (G[W/V] \wedge \lfloor\lfloor W = V \rfloor\rfloor \wedge (\overrightarrow{W} = \overrightarrow{V}))$$

Proof. From ($\exists$-1) and ($\exists$-2). $\square$

3.5. Chopping points

Definition 3.7 (Super-dense chop)
Let V be the state variables used in F and G. Define

$$\begin{aligned} F \circ G \;=_{df}\; & \exists x, V_F, V_G \bullet \\ & (F[V_F/V] \;\wedge\; [\![V = V_F]\!] \;\wedge\; (\overleftarrow{V}{=}\overleftarrow{V_F}) \;\wedge\; (\overrightarrow{V_F}{=}\, x))\,\hat{}\, \\ & (G[V_G/V] \;\wedge\; [\![V = V_G]\!] \;\wedge\; (\overrightarrow{V}{=}\overrightarrow{V_G}) \;\wedge\; (\overleftarrow{V_G}{=}\, x)) \qquad \square \end{aligned}$$

The chop operator $\hat{}$ is used to compose the continuously evolving hybrid systems [ZHR91], whereas the relational composition operator is used to model the sequential composition of imperative programming languages [HoH98, Mor94]. The following theorem states that $\circ$ can be seen as the product of the chop operator $\hat{}$ and the relational composition operator,

Theorem 3.8
If $\overleftarrow{V}$ and $\overrightarrow{V}$ do not occur in F nor G, then

$$(F \wedge p(\overleftarrow{V}, \overrightarrow{V})) \circ (q(\overleftarrow{V}, \overrightarrow{V}) \wedge G) \;=\; (F\,\hat{}\,G) \;\wedge\; \exists x \bullet (p(\overleftarrow{V}, x) \wedge q(x, \overrightarrow{V}))$$

Theorem 3.9 ($\circ$ and $\hat{}$)
If $\overrightarrow{V} \notin F$ and $\overleftarrow{V} \notin G$, then $F \circ G \;=\; F\,\hat{}\,G$.

Proof.

$$\begin{aligned} & F \circ G && \{\text{definition}\} \\ = \;& \exists x, V_F, V_G \bullet \\ & \quad (F[V_F/V] \;\wedge\; [\![V = V_F]\!] \;\wedge\; (\overleftarrow{V}{=}\overleftarrow{V_F}) \;\wedge\; (\overrightarrow{V_F}{=}\, x))\,\hat{}\, \\ & \quad (G[V_G/V] \;\wedge\; [\![V = V_G]\!] \;\wedge\; (\overrightarrow{V}{=}\overrightarrow{V_G}) \;\wedge\; (\overleftarrow{V_G}{=}\, x)) && \{(\exists\text{-}2)\} \\ = \;& \exists x \bullet (\exists V_1 \bullet (F[V_1/V] \wedge [\![V = V_1]\!] \wedge (\overleftarrow{V}{=}\overleftarrow{V_1})) \;\wedge\; (\exists V_1 \bullet (\overrightarrow{V_1}{=}\, x)))\,\hat{}\, \\ & \quad (\exists V_2 \bullet (G[V_2/V] \wedge [\![V = V_2]\!] \wedge (\overrightarrow{V}{=}\overrightarrow{V_2})) \;\wedge\; (\exists V_2 \bullet (\overleftarrow{V_2}{=}\, x))) \\ & && \{\text{Coro of } (\exists\text{-}4)\} \\ = \;& \exists x \bullet (F\,\hat{}\,G) && \{x \text{ not used by } F \text{ and } G\} \\ = \;& F\,\hat{}\,G && \square \end{aligned}$$

$\circ$ also enjoys the following familiar algebraic laws:

Theorem 3.10
(1) $(F \circ G) \circ H \;=\; F \circ (G \circ H)$

(2) $F \circ I \;=\; F \;=\; I \circ F$, where $I \;=_{df}\; (\ell = 0) \;\wedge\; (\overrightarrow{V}{=}\overleftarrow{V})$

(3) $F \circ (G \vee H) \;=\; (F \circ G) \vee (F \circ H)$

$$(G \vee H) \circ F \;=\; (G \circ F) \vee (H \circ F)$$

(4) $F \circ \mathbf{false} \;=\; \mathbf{false} \;=\; \mathbf{false} \circ G$

(5) $(p(\overleftarrow{V}) \wedge F) \circ G \;=\; p(\overleftarrow{V}) \wedge (F \circ G)$

(6) $F \circ (G \wedge q(\overrightarrow{V})) \;=\; (F \circ G) \wedge q(\overrightarrow{V})$

(7) $(F \wedge r(\overrightarrow{V})) \circ G \;=\; F \circ (r(\overleftarrow{V}) \wedge G)$

(8) If $\overrightarrow{V} \notin F$ and $\overleftarrow{V} \notin G$, then

$$(F \wedge (\overrightarrow{V} = y)) \circ G \;=\; F \frown G$$

$$F \circ ((\overleftarrow{V} = y) \wedge G) \;=\; F \frown G$$

$$(F \wedge (\overrightarrow{V} = y)) \circ ((\overleftarrow{V} = z) \wedge G) \;=\; (F \frown G) \wedge (y = z)$$

3.6. Fixed Points

Definition 3.11 (Greatest lower bound)
Let $\mathcal{F}$ be a set of formulae. We define its greatest lower bound $\sqcap\mathcal{F}$ by

$$\mathcal{M}(\sqcap\mathcal{F})(\sigma) \;=_{df}\; \exists F \in \mathcal{F} \bullet \mathcal{M}(F)(\sigma) \qquad \square$$

$\sqcap$ can be defined algebraically by the following law:

(glb-1) $G \Leftarrow \sqcap\mathcal{F}$ iff $G \Leftarrow F$ for all $F \in \mathcal{F}$.

Corollary
(1) $\sqcap\emptyset \;=\; \mathbf{false}$

(2) $\sqcap(\mathcal{F}1 \cup \mathcal{F}2) \;=\; (\sqcap\mathcal{F}1) \;\vee\; (\sqcap\mathcal{F}2)$

(3) $\sqcap\{F\} \;=\; F$ $\square$

The greatest lower bound operator distributes over both $\frown$ and $\circ$.

(glb-2) Let $\star \in \{\frown, \circ\}$

$$(\sqcap\mathcal{F}) \star G \;=\; \sqcap\,\{F \star G \mid F \in \mathcal{F}\}$$

$$G \star (\sqcap\mathcal{F}) \;=\; \sqcap\,\{G \star F \mid F \in \mathcal{F}\}$$

Definition 3.12 (Weakest fixed point)
Let Φ be a monotonic mapping of formulae. We define its weakest fixed point by

$$\mu X \bullet \Phi(X) \;=_{df}\; \sqcap\,\{F \mid F \Rightarrow \Phi(F)\} \qquad \square$$

From Tarski's Fixed Point Theorem [Tar55] it follows that $\mu X \bullet \Phi(X)$ is subject to the following laws.

(μ-1) $\quad \Phi(\mu X \bullet \Phi(X)) \;=\; \mu X \bullet \Phi(X)$

(μ-2) If $F \Rightarrow \Phi(F)$, then $F \Rightarrow \mu X \bullet \Phi(X)$.

Definition 3.13 (Least upper bound)
Let $\mathcal{F}$ be a set of formulae. We define its least upper bound $\sqcup\mathcal{F}$ by

$$\mathcal{M}(\sqcup\mathcal{F})(\sigma) \;=_{df}\; \forall F \in \mathcal{F} \bullet \mathcal{M}(F)(\sigma) \qquad \square$$

$\sqcup$ can be defined algebraically by the following law:

(lub-1) $G \Rightarrow \sqcup\mathcal{F}$ iff $G \Rightarrow F$ for all $F \in \mathcal{F}$.

Corollary
(1) $\sqcup\emptyset \;=\; \mathbf{true}$
(2) $\sqcup(\mathcal{F}1 \cup \mathcal{F}2) \;=\; (\sqcup\mathcal{F}1) \;\wedge\; (\sqcup\mathcal{F}2)$
(3) $\sqcup\{F\} \;=\; F$ $\square$

4. Link the Untimed Refinement Calculus with the Timed Calculus

The meaning of a program of the Guarded Command Language [Dij76] is defined in [HoH98] by a predicate

$$pre(v) \vdash post(v,\, v') \;=_{df}\; (ok \wedge pre(v)) \Rightarrow (ok' \wedge post(v,\, v'))$$

where

- Both ok and ok' are Boolean variables, where ok records the observation that the program has been started, and ok' records the observation that the program has terminated.
- v and v' respectively denote the initial and final values of the program variables v.
- The predicate $pre(v)$ is an assumption which the program can rely on when the program is initiated.
- The predicate $post(v,\, v')$ is a commitment which must be true when the program terminates.

A *design* is a relation whose predicate is expressed in this form. We will use D to range over the set of designs in the remainder of this section. This is the approach used in the refinement calculus [HoH98, Mor94]. Designs can be combined by disjunction $\vee$, conditional $\lhd b \rhd$ and relational composition.

Definition 4.1 (Relational composition)

$$Q(s'); R(s) \;=_{df}\; \exists m \bullet (Q(m) \wedge R(m))$$

Theorem 4.2 (Laws of designs) [HoH98]
(1) $(b_1 \vdash R_1) \vee (b_2 \vdash R_2) = (b_1 \wedge b_2) \vdash (R_1 \vee R_2)$
(2) $(b_1 \vdash R_1) \lhd c \rhd (b_2 \vdash R_2) = (b_1 \lhd c \rhd b_2) \vdash (R_1 \lhd c \rhd R_2)$
(3) $(b_1 \vdash R_1); (b_2 \vdash R_2) = (b_1 \wedge \neg(R_1; \neg b_2)) \vdash (R_1; R_2)$ $\square$

We next establish links between the untimed refinement calculus [Jon90, Mor94] and the timed one.

Definition 4.3 (Mapping designs to DC formulae)

Let D be a design. We define its timed interpretation by

$$\mathcal{T}(D) \;=_{df}\; \exists v, v' \bullet ((\overleftarrow{V} = v) \;\wedge\; D \;\wedge\; (v' = \overrightarrow{V})) \qquad \square$$

Example 4.4

$$\mathcal{T}(\mathbf{true} \vdash (v' = v + 1)) \;=\; ok \Rightarrow (ok' \wedge (\overrightarrow{V} = \overleftarrow{V} + 1)) \qquad \square$$

Theorem 4.5

$\mathcal{T}$ is injective, and

$$\mathcal{T}(pre(v) \vdash post(v, v')) \;=\; (ok \wedge pre(\overleftarrow{V})) \Rightarrow (ok' \wedge post(\overleftarrow{V}, \overrightarrow{V})) \qquad \square$$

Definition 4.6 (Sequential composition of DC formulae)

Let $R1$ and $R2$ be formulae with global variables ok and ok'. We define their *sequential composition* $R1; R2$ by

$$R1; R2 \;=_{df}\; \exists o \bullet (R1[o/ok'] \circ R2[o/ok]) \qquad \square$$

Definition 4.7 (Relational formulae)

A DC formula is *relational* if it can be expressed in the form

$$P(\overleftarrow{V}) \vdash Q(\overleftarrow{V}, \overrightarrow{V}) \;=_{df}\; (ok \wedge P(\overleftarrow{V})) \Rightarrow (ok' \wedge Q(\overleftarrow{V}, \overrightarrow{V})) \qquad \square$$

Theorem 4.8 (Closure of relational formulae)

If R_1 and R_2 are relational formulae, so are $R_1 \vee R_2$, $R_1 \lhd b(\overleftarrow{V}) \rhd R_2$ and $R_1; R_2$. $\square$

The conclusion for disjunction generalises to an arbitrary set of relational formulae, and a similar law hold for arbitrary intersection.

Theorem 4.9 (Closure of glb and lub)

(1) $\sqcap (P_i \vdash Q_i) \;=\; (\sqcup P_i) \vdash (\sqcap Q_i)$

(2) $\sqcup (P_i \vdash Q_i) \;=\; (\sqcap P_i) \vdash (\sqcup (P_i \Rightarrow Q_i))$ $\square$

This means that relational formulae form a complete lattice under implication ordering. It contains a bottom element **true** and a top element $\neg ok$.

Lemma 4.10 (Link between weakest fixed points) [Mat95]

If f is universally conjunctive, i.e., for any set S,

$$f(\sqcup S) \;=\; \sqcup \{f(X) \mid X \in S\}$$

and g and h are monotonic functions on complete lattices satisfying $f \circ g = h \circ f$, then $f(\mu g) = \mu h$. $\square$

Finally we can prove $\mathcal{T}$ is a homomorphism.

Theorem 4.11

(1) $\mathcal{T}(D_1 \vee D_2) \;=\; \mathcal{T}(D_1) \vee \mathcal{T}(D_2)$

(2) $\mathcal{T}(D_1 \wedge D_2) \;=\; \mathcal{T}(D_1) \wedge \mathcal{T}(D_2)$

(3) $\mathcal{T}(D_1 \lhd b(v) \rhd D_2) \;=\; \mathcal{T}(D_1) \lhd b(\overleftarrow{V}) \rhd \mathcal{T}(D_2)$

(4) $\mathcal{T}(D_1; D_2) \;=\; \mathcal{T}(D_1); \mathcal{T}(D_2)$

(5) $\mathcal{T}(\mu X \bullet ((D; X) \lhd b(v) \rhd \Pi)) \;=\; \mu X \bullet ((\mathcal{T}(D); X) \lhd b(\overleftarrow{V}) \rhd \mathcal{T}(\Pi))$

Proof. (4) Let $D_i \;=_{df}\; pre_i(v) \vdash post_i(v,\, v')$ for $i = 1,\, 2$.

$$
\begin{array}{lll}
 & \mathcal{T}(D_1); \mathcal{T}(D_2) & \{\text{Theorem 4.5}\} \\
= & (pre_1(\overleftarrow{V}) \vdash post_1(\overleftarrow{V},\, \overrightarrow{V})); (pre_2(\overleftarrow{V}) \vdash post_2(\overleftarrow{V},\, \overrightarrow{V})) & \{\text{Theorem 3.10}\} \\
= & (pre_1(\overleftarrow{V}) \wedge \neg(post_2(\overleftarrow{V},\, \overrightarrow{V}) \circ \neg pre_2(\overleftarrow{V}))) & \\
 & \vdash (post_1(\overleftarrow{V},\, \overrightarrow{V}) \circ post_2(\overleftarrow{V},\, \overrightarrow{V})) & \{\text{Theorem 3.8}\} \\
= & (pre_1(\overleftarrow{V}) \wedge \neg\exists x \bullet (post_2(\overleftarrow{V},\, x) \wedge \neg pre_2(x))) & \\
 & \vdash \exists y \bullet (post_1(\overleftarrow{V},\, y) \wedge post_2(y,\, \overrightarrow{V})) & \{\text{Theorem 4.5}\} \\
= & \mathcal{T}((pre_1(v) \wedge \neg\exists x \bullet (post_2(v,\, x) \wedge \neg pre_2(x))) & \\
 & \vdash \exists y \bullet (post_1(v,\, y) \wedge post_2(y,\, v'))) & \{\text{Theorem 4.2}\} \\
= & \mathcal{T}(D_1; D_2) & \square
\end{array}
$$

(5) Let $g(X) \;=\; (D; X) \lhd b(v) \rhd \Pi$, and $h(X) \;=\; (\mathcal{T}(D); X) \lhd b(\overleftarrow{V}) \rhd \mathcal{T}(\Pi)$. The conclusion follows from Lemma 4.10 and the fact that $\mathcal{T}$ is universally conjunctive, and $\mathcal{T} \circ g = h \circ \mathcal{T}$. $\square$

Remark

An alternative timed interpretation of a design $pre \vdash post$ can be

$$\mathcal{A}(pre \vdash post) \;=_{df}\; pre(\overleftarrow{V}) \vdash ((\ell = 0) \wedge post(\overleftarrow{V},\, \overrightarrow{V}))$$

where execution of a termination program does not take time. Like $\mathcal{T}$, $\mathcal{A}$ is also a homomorphism.

Clearly, the theory of relational formulae is less able to describe things accurately than DC that contains more formulae. But for any DC formula F, we can single out those relational formulae that *approximate* to D. The *best* approximation to P is the least upper bound of this set.

Definition 4.12 (Approximation)

$$\mathbf{approx}(F) \;=_{df}\; \sqcup \{P \vdash Q \mid F \Rightarrow (P \vdash Q)\}$$

From Theorem 4.9(2) it follows that $\mathbf{approx}(F)$ is relational. $\square$

Theorem 4.13

approx is monotonic and idempotent, and satisfies

$$F \Rightarrow \mathbf{approx}(F) \qquad \square$$

Theorem 4.14

(1) $\mathbf{approx}(F_1 \vee F_2) \;\Rightarrow\; \mathbf{approx}(F_1) \vee \mathbf{approx}(F_2)$

(2) $\mathbf{approx}(F_1 \lhd b(\overleftarrow{V}) \rhd F_2) \Rightarrow (\mathbf{approx}(F_1) \lhd b(\overleftarrow{V}) \rhd \mathbf{approx}(F_2))$

(3) $\mathbf{approx}(F_1; F_2) \;\Rightarrow\; \mathbf{approx}(F_1); \mathbf{approx}(F_2)$ $\square$

Theorem 4.15 (Closure of weakest fixed point)

If Φ is formed by relational formulae and operators $\vee$, $\lhd b \rhd$ and ;, then $\mu X \bullet \Phi(X)$ is also a relational formula.

Proof.

$$\begin{array}{lll} & \mathbf{approx}(\mu X \bullet \Phi(X)) & \{\text{fixed point}\} \\ = & \mathbf{approx}(\Phi(\mu X \bullet \Phi(X))) & \{\text{Theorem 4.14}\} \\ \Rightarrow & \Phi(\mathbf{approx}(\mu X \bullet \Phi(X))) & \end{array}$$

from which and the law of weakest fixed point it follows that

$$\mathbf{approx}(\mu X \bullet \Phi(X)) \Rightarrow \mu X \bullet \Phi(X)$$

The opposite inequation follows from Theorem 4.13. $\square$

5. Sequential Hybrid Programs

5.1. Syntax and Semantics

In this section, we study Sequential Hybrid Programs (SHP) initially proposed in [Xu97, PWX98]. As a reactive system, an SHP program may well be nonterminating. We now not only observe the initial and the final states, but also the intermediate behaviours of the system. As before, we use Boolean variable ok to denote that the program has been started. But instead of termination, ok' now denotes that the program is stable, i.e., the program is not in an endless loop of instantaneous transitions. To distinguish intermediate behaviours from those on termination, we introduce another two global variables $wait, wait' : Bool$. When $wait$ is true the program is started in an intermediate state, and when $wait'$ is true the program has not terminated.

Taking new global variables $wait$ and $wait'$ into account, Definition 4.6 of sequential composition is generalised as follows

Definition 5.1. (Revised definition of sequential composition)

$$P; Q \;=_{df}\; \exists w, o \bullet \; (P[w, o/wait', ok'] \circ Q[w, o/wait, ok]) \qquad \square$$

Sequential composition has the unit

$$II \;=_{df}\; ok \Rightarrow \; (ok' \;\wedge\; (\ell = 0) \;\wedge\; (\overleftarrow{V} = \overrightarrow{V}) \wedge (wait' = wait))$$

The introduction of intermediate waiting state has implication for sequential composition; all the intermediate observation of P are also intermediate observations of $(P;Q)$. If Q is asked to start in a waiting state of P, it leaves the state unchanged, i.e., it satisfies the *healthiness condition*

(H) $Q \;=\; II \lhd wait \rhd Q$

Theorem 5.2. (Closure of healthy formulae)

If both P_1 and P_2 are healthy formulae, so are $(P_1 \vee P_2)$, $P_1 \lhd b(\overleftarrow{V}) \rhd P_2$, and $(P_1;P_2)$.

Proof. we only show the case for sequential composition.

$$\begin{array}{lll} & (P_1;P_2) & \{(H)\} \\ = & (II \lhd wait \rhd P_1);P_2 & \{\text{Theorem 3.10(3)}\} \\ = & (II;P_2) \lhd wait \rhd (P_1;P_2) & \{II;X \;=X\} \\ = & P_2 \lhd wait \rhd (P_1;P_2) & \{(H)\} \\ = & (II \lhd wait \rhd P_2) \lhd wait \rhd (P_1;P_2) & \{\text{Def of } \lhd b \rhd\} \\ = & II \lhd wait \rhd (P_1;P_2) & \square \end{array}$$

We denote the set of all healthy formulae by HD.

Theorem 5.3. HD forms a complete lattice.

Proof. Q is healthy iff it satisfies

$$Q \;=\; \mathcal{H}(Q)$$

where

$$\mathcal{H}(X) \;=_{df}\; II \lhd wait \rhd X$$

The conclusion follows from Tarski's Theory [Tar55]. $\square$

For a mapping F closed in HD, i.e., $\phi \in HD$ implies $F(\phi) \in HD$, we use $\mu_{HD} \bullet F$ to stand for the weakest fixed point in the complete lattice HD.

We next study SHP using DC formulae satisfying the above healthiness condition. The language of SHP can be considered as an extension of the refinement calculus of Section 4 by a specification statement using DC formulae for describing possibly hybrid components with time dependent dynamics. To express the programs more conveniently, we introduce a syntax which is similar to conventional programming notations. For simplicity, we assume v is the only system variable. It is straightforward to extend to the case where there is a vector of variables.

$$P ::= S \mid P_1;P_2 \mid P_1 \sqcap P_2 \mid (P_1 \lhd b \rhd P_2) \mid \mathsf{while}\ b\ \mathsf{do}\ P\ \mathsf{od}.$$

Denoting a design of an SHP program, S is of the form $(\alpha \vdash_{\mathcal{H}} (\beta \lhd wait' \rhd \gamma))$ where α, β and γ are DC formulae which do not mention ok, ok', $wait$ and $wait'$; Sequential composition is represented by $P_1;P_2$ and its meaning is that if P_1 terminates then P_2 is executed immediately afterwards; $\sqcap$ is the nondeterministic choice; Iteration statement is executed repeatedly until the Boolean guard becomes false.

An SHP design is interpreted as a DC formula

Definition 5.4.

$$(\alpha \vdash_{\mathcal{H}} (\beta \lhd wait' \rhd \gamma)) \quad =_{df} \quad \mathcal{H}((ok \wedge \alpha) \Rightarrow (ok' \wedge (\beta \lhd wait' \rhd \gamma)))$$

The meaning is that if the program is started, then under the condition α, it guarantees that it will not diverge and the intermediate behaviours satisfy β while terminated behaviours satisfy γ. □

A particular form of SHP designs describes the dynamics that a phase of a hybrid system satisfies. We call this a phase statement, denoted by $\ll C \gg$ where C is a DC formula. Continuous evolution can be specified by an invariant expressing the dynamical laws. Formally,

$$[\![\ll C \gg]\!] \ =_{df} \ (\mathbf{true} \vdash_{\mathcal{H}} C)$$

The semantics of SHP program combinators is defined as usual

$$\begin{array}{rcl}
[\![P_1;P_2]\!] & =_{df} & [\![P_1]\!];[\![P_2]\!], \\
[\![P_1 \sqcap P_2]\!] & =_{df} & [\![P_1]\!] \vee [\![P_2]\!], \\
[\![(P_1 \lhd b \rhd P_2)]\!] & =_{df} & ([\![P_1]\!] \lhd b(\overleftarrow{V}) \rhd [\![P_2]\!]), \\
[\![\mathsf{while}\ b\ \mathsf{do}\ P\ \mathsf{od}]\!] & =_{df} & \mu_{HD} X \bullet (([\![P]\!]; X) \lhd b(\overleftarrow{V}) \rhd II).
\end{array}$$

As in Section 4, we map a design in refinement calculus, henceforth called a sequential design to distinguish from a design in SHP, into a DC formula

Definition 5.5. (Revised interpretation of sequential design)

$$\mathcal{R}(D) \ =_{df} \ \mathcal{H}(\mathcal{A}(D); (ok \Rightarrow (ok' \wedge \neg wait' \wedge II)))$$

where $\mathcal{A}(D)$ is the alternative interpretation of designs in Section 4. We take this interpretation because in SHP we express timing behaviours by DC formulae and assume other statements do not take time. It is easy to prove that

$$\mathcal{R}(b \vdash R) \ = \ II \lhd wait \rhd ((ok \wedge b(\overleftarrow{V})) \Rightarrow (ok' \wedge \neg wait' \wedge R(\overleftarrow{V}, \overrightarrow{V}) \wedge (\ell = 0))) \square$$

It follows that $\mathcal{R}(D)$ is an SHP design. The following theorem states $\mathcal{R}$ is also a homomorphism over sequential designs

Theorem 5.6.

(1) $\mathcal{R}(D_1 \vee D_2) \ = \ \mathcal{R}(D_1) \vee \mathcal{R}(D_2)$

(2) $\mathcal{R}(D_1 \wedge D_2) \ = \ \mathcal{R}(D_1) \wedge \mathcal{R}(D_2)$

(3) $\mathcal{R}(D_1 \lhd b(v) \rhd D_2) \ = \ \mathcal{R}(D_1) \lhd b(\overleftarrow{V}) \rhd \mathcal{R}(D_2)$

(4) $\mathcal{R}(D_1; D_2) \ = \ \mathcal{R}(D_1); \mathcal{R}(D_2)$

(5) $\mathcal{R}(\mu X \bullet ((D; X) \lhd b(v) \rhd II)) \ = \ \mu_{HD} X \bullet ((\mathcal{R}(D); X) \lhd b(\overleftarrow{V}) \rhd II)$ □

Theorem 5.7. If $D_1 \Rightarrow D_2$ holds in sequential refinement calculus, then $\mathcal{R}(D_1) \Rightarrow \mathcal{R}(D_2)$. □

This together with Theorem 5.6 indicates that laws of sequential refinement calculus are still valid when sequential designs are considered as SHP programs.

Example 5.8.

$x := x+1; x := x+2 \;=\; x := x+3$
$x := e; (P \lhd b(x) \rhd Q) \;=\; ((x := e; P) \lhd b(e) \rhd (x := e; Q))$ □

We can define the Hoare triple as

$$\{pre(v)\}\ D\ \{post(v)\} \;=_{df}\; D \Rightarrow (pre(v) \vdash post(v')).$$

Theorem 5.7 also says that one can still use the traditional Hoare logic to verify the discrete component of a SHP program.

5.2. Verification of SHP Programs

Refinement laws expressed in Theorem 4.2 can be generalised to SHP designs

Theorem 5.9.

1. $(\alpha_1 \vdash_{\mathcal{H}} (\beta_1 \lhd wait' \rhd \gamma_1)) \;\vee\; (\alpha_2 \vdash_{\mathcal{H}} (\beta_2 \lhd wait' \rhd \gamma_2))$
   $= (\alpha_1 \wedge \alpha_2 \vdash_{\mathcal{H}} ((\beta_1 \vee \beta_2) \lhd wait' \rhd (\gamma_1 \vee \gamma_2)))$,
2. $((\alpha_1 \vdash_{\mathcal{H}} (\beta_1 \lhd wait' \rhd \gamma_1)) \lhd b(\overleftarrow{V}) \rhd (\alpha_2 \vdash_{\mathcal{H}} (\beta_2 \lhd wait' \rhd \gamma_2)))$
   $= ((\alpha_1 \lhd b(\overleftarrow{V}) \rhd \alpha_2) \vdash_{\mathcal{H}} ((\beta_1 \lhd b(\overleftarrow{V}) \rhd \beta_2) \lhd wait' \rhd (\gamma_1 \lhd b(\overleftarrow{V}) \rhd \gamma_2)))$,
3. $(\alpha_1 \vdash_{\mathcal{H}} (\beta_1 \lhd wait' \rhd \gamma_1))\ ;\ (\alpha_2 \vdash_{\mathcal{H}} (\beta_2 \lhd wait' \rhd \gamma_2))$
   $= (\neg(\neg\alpha_1; \mathbf{true}) \wedge \neg(\gamma_1; \neg\alpha_2) \vdash_{\mathcal{H}} ((\beta_1 \vee (\gamma_1; \beta_2)) \lhd wait' \rhd (\gamma_1; \gamma_2)))$. □

We denote by $\{\alpha\}\ P\ \{(\beta \lhd wait' \rhd \gamma)\}$ that an SHP program P implements a design $(\alpha \vdash_{\mathcal{H}} (\beta \lhd wait' \rhd \gamma))$. This is interpreted as

$$[\![P]\!] \Rightarrow (\alpha \vdash_{\mathcal{H}} (\beta \lhd wait' \rhd \gamma)).$$

The following proof rules are derived directly from Theorem 5.9.

$$\frac{\{\alpha\}\ P_1\ \{(\beta \lhd wait' \rhd \gamma)\}, \quad \{\alpha\}\ P_2\ \{(\beta \lhd wait' \rhd \gamma)\}}{\{\alpha\}\ P_1 \sqcap P_2\ \{(\beta \lhd wait' \rhd \gamma)\}}$$

$$\frac{\{\alpha \wedge b(\overleftarrow{V})\}\ P_1\ \{(\beta \lhd wait' \rhd \gamma)\}, \quad \{\alpha \wedge \neg b(\overleftarrow{V})\}\ P_2\ \{(\beta \lhd wait' \rhd \gamma)\}}{\{\alpha\}\ (P_1 \lhd b \rhd P_2)\ \{(\beta \lhd wait' \rhd \gamma)\}}$$

$$\frac{\alpha_2 \Rightarrow \alpha_1,\ \beta_1 \Rightarrow \beta_2,\ \gamma_1 \Rightarrow \gamma_2, \quad \{\alpha_1\}\ P\ \{(\beta_1 \lhd wait' \rhd \gamma_1)\}}{\{\alpha_2\}\ P\ \{(\beta_2 \lhd wait' \rhd \gamma_2)\}}$$

In the following, we study several special cases of SHP design formulae and investigate verification rules for them.

Special case 1: formula α in the SHP design is a state predicate $pre(\overleftarrow{V})$, denoting a precondition.

$$\frac{\begin{array}{c}\{pre(\overleftarrow{V})\}\ P_1\ \{(\beta_1 \lhd wait' \rhd (\gamma_1 \wedge mid(\overrightarrow{V})))\} \\ \{mid(\overleftarrow{V})\}\ P_2\ \{(\beta_2 \lhd wait' \rhd (\gamma_2 \wedge post(\overrightarrow{V})))\}\end{array}}{\{pre(\overleftarrow{V})\}\ P_1; P_2\ \{((\beta_1 \vee (\gamma_1; \beta_2)) \lhd wait' \rhd ((\gamma_1; \gamma_2) \wedge post(\overrightarrow{V})))\}}$$

Special case 2: the SHP design is of the form $(\mathit{pre}(\overleftarrow{V}) \vdash_{\mathcal{H}} (\alpha \lhd \mathit{wait}' \rhd (\alpha \wedge \mathit{post}(\overrightarrow{V}))))$. Denote $\{\mathit{pre}(\overleftarrow{V})\}\ \ P\ \ \{(\alpha \lhd \mathit{wait}' \rhd (\alpha \wedge \mathit{post}(\overrightarrow{V})))\}$ by $\alpha:\ \{\mathit{pre}(\overleftarrow{V})\}\ \ P\ \ \{\mathit{post}(\overrightarrow{V})\}$.

$$\frac{\alpha:\ \{\mathit{pre}(\overleftarrow{V})\}\ \ P_1\ \ \{\mathit{mid}(\overrightarrow{V})\},\qquad \beta:\ \{\mathit{mid}(\overleftarrow{V})\}\ \ P_2\ \ \{\mathit{post}(\overrightarrow{V})\}}{\{\mathit{pre}(\overleftarrow{V})\}\ \ P_1;P_2\ \ \{((\alpha \vee (\alpha;\beta)) \lhd \mathit{wait}' \rhd ((\alpha;\beta) \wedge \mathit{post}(\overrightarrow{V})))\}}$$

$$\frac{\begin{array}{c}\alpha;\beta\ \Rightarrow\ \alpha\\ \alpha:\ \{\mathit{pre}(\overleftarrow{V})\}\ \ P_1\ \ \{\mathit{mid}(\overrightarrow{V})\},\qquad \beta:\ \{\mathit{mid}(\overleftarrow{V})\}\ \ P_2\ \ \{\mathit{post}(\overrightarrow{V})\}\end{array}}{\alpha:\ \{\mathit{pre}(\overleftarrow{V})\}\ \ P_1;P_2\ \ \{\mathit{post}(\overrightarrow{V})\}}$$

$$\frac{\begin{array}{c}\alpha;\alpha\ \Rightarrow\ \alpha,\qquad \ell = 0\ \Rightarrow\ \alpha\\ \{(b \wedge r)(\overleftarrow{V})\}\ \ P\ \ \{(\alpha \lhd \mathit{wait}' \rhd (\alpha \wedge \ell > c \wedge r(\overrightarrow{V})))\}\end{array}}{\alpha:\ \{r(\overleftarrow{V})\}\ \ \mathsf{while}\ \ b\ \ \mathsf{do}\ \ P\ \ \mathsf{od}\ \ \{r(\overrightarrow{V})\}}$$

where c is a positive constant.

Proof. We only prove the last rule. First, we show

$(\neg \mathit{wait} \wedge \mathit{ok} \wedge r(\overleftarrow{V}) \wedge \llbracket\, \mathsf{while}\ \ b\ \ \mathsf{do}\ \ P\ \ \mathsf{od}\, \rrbracket) \Rightarrow (\mathit{ok}' \wedge (\alpha \lhd \mathit{wait}' \rhd (\alpha \wedge r(\overrightarrow{V}))))$

Let $\llbracket\, W \,\rrbracket\ =_{df}\ \llbracket\, \mathsf{while}\ \ b\ \ \mathsf{do}\ \ P\ \ \mathsf{od}\, \rrbracket$ in the proof below.

$$\begin{array}{ll}
\quad \neg \mathit{wait} \wedge \mathit{ok} \wedge r(\overleftarrow{V}) \wedge \llbracket W \rrbracket & \{\text{fixed point}\}\\
= \neg \mathit{wait} \wedge \mathit{ok} \wedge r(\overleftarrow{V}) \wedge ((\llbracket P \rrbracket; \llbracket W \rrbracket) \lhd b(\overleftarrow{V}) \rhd \mathit{II}) & \{\text{Def of } (\lhd b \rhd)\}\\
\Rightarrow ((\neg \mathit{wait} \wedge \mathit{ok} \wedge (b \wedge r)(\overleftarrow{V}) \wedge \llbracket P \rrbracket); \llbracket W \rrbracket)\ \vee\ (\neg \mathit{wait} \wedge \mathit{ok} \wedge r(\overleftarrow{V}) \wedge \mathit{II}) & \\
 & \{\text{premises of the rule and Def of } \mathit{II}\}\\
\Rightarrow ((\neg \mathit{wait} \wedge \mathit{ok} \wedge (b \wedge r)(\overleftarrow{V}) & \\
\qquad \wedge ((b \wedge r)(\overleftarrow{V}) \vdash_{\mathcal{H}} (\alpha \lhd \mathit{wait}' \rhd \alpha \wedge \ell > c \wedge r(\overrightarrow{V})))); \llbracket W \rrbracket) & \\
\quad \vee\ (\mathit{ok}' \wedge \neg \mathit{wait}' \wedge r(\overrightarrow{V}) \wedge \ell = 0) & \{\text{Def of } \vdash_{\mathcal{H}} \text{ and } \mathcal{H}\}\\
\Rightarrow ((\neg \mathit{wait} \wedge \mathit{ok} \wedge (b \wedge r)(\overleftarrow{V}) & \\
\qquad \wedge ((\mathit{ok} \wedge (b \wedge r)(\overleftarrow{V})) \Rightarrow (\mathit{ok}' \wedge (\alpha \lhd \mathit{wait}' \rhd \alpha \wedge \ell > c \wedge r(\overrightarrow{V}))))); \llbracket W \rrbracket) & \\
\quad \vee\ (\mathit{ok}' \wedge \neg \mathit{wait}' \wedge r(\overrightarrow{V}) \wedge \alpha) & \{\text{predicate calculus}\}\\
\Rightarrow ((\mathit{ok}' \wedge (\alpha \lhd \mathit{wait}' \rhd \alpha \wedge \ell > c \wedge r(\overrightarrow{V}))); \llbracket W \rrbracket) & \\
\quad \vee\ (\mathit{ok}' \wedge \neg \mathit{wait}' \wedge r(\overrightarrow{V}) \wedge \alpha) & \{\text{Def of } (\lhd \mathit{wait}' \rhd)\}\\
= ((\mathit{ok}' \wedge \mathit{wait}' \wedge \alpha); \llbracket W \rrbracket)\ \vee\ ((\mathit{ok}' \wedge \neg \mathit{wait}' \wedge \alpha \wedge \ell > c \wedge r(\overrightarrow{V})); \llbracket W \rrbracket) & \\
\quad \vee\ (\mathit{ok}' \wedge \neg \mathit{wait}' \wedge r(\overrightarrow{V}) \wedge \alpha) & \{\llbracket W \rrbracket \text{ is healthy}\}\\
\Rightarrow ((\mathit{ok}' \wedge \mathit{wait}' \wedge \alpha); \mathit{II})\ \vee\ (\mathit{ok}' \wedge \neg \mathit{wait}' \wedge r(\overrightarrow{V}) \wedge \alpha) & \\
\quad \vee\ ((\alpha \wedge \ell > c); (\neg \mathit{wait} \wedge \mathit{ok} \wedge r(\overleftarrow{V}) \wedge \llbracket W \rrbracket)) & \{\text{Def of } (\lhd \mathit{wait}' \rhd)\}\\
\Rightarrow (\mathit{ok}' \wedge (\alpha \lhd \mathit{wait}' \rhd (\alpha \wedge r(\overrightarrow{V}))))\ \vee\ ((\alpha \wedge \ell > c); (\neg \mathit{wait} \wedge \mathit{ok} \wedge r(\overleftarrow{V}) \wedge \llbracket W \rrbracket)) &
\end{array}$$

By induction, it follows that

$$
\begin{aligned}
& \neg wait \wedge ok \wedge r(\overleftarrow{V}) \wedge [\![W]\!] \\
\Rightarrow & \forall n \bullet ((ok' \wedge (\alpha \lhd wait' \rhd (\alpha \wedge r(\overrightarrow{V})))) \\
& \qquad \vee ((\alpha \wedge \ell > nc); (\neg wait \wedge ok \wedge r(\overleftarrow{V}) \wedge [\![W]\!]))) \\
= & (ok' \wedge (\alpha \lhd wait' \rhd (\alpha \wedge r(\overrightarrow{V})))) \\
& \vee ((\alpha \wedge (\forall n \bullet \ell > nc)); (\neg wait \wedge ok \wedge r(\overleftarrow{V}) \wedge [\![W]\!]))
\end{aligned}
$$

Since $(\forall n \bullet \ell > nc) = \mathbf{false}$, we have

$$\neg wait \wedge ok \wedge r(\overleftarrow{V}) \wedge [\![W]\!] \Rightarrow (ok' \wedge (\alpha \lhd wait' \rhd (\alpha \wedge r(\overrightarrow{V}))))$$

Therefore,

$$
\begin{aligned}
& [\![W]\!] && \{\text{predicate calculus}\} \\
\Rightarrow & (wait \wedge [\![W]\!]) \ \vee \ (\neg wait \wedge [\![W]\!]) && \{\text{previous result}\} \\
\Rightarrow & (wait \wedge II) \ \vee \ (\neg wait \wedge ((ok \wedge r(\overleftarrow{V}) \Rightarrow (ok' \wedge (\alpha \lhd wait' \rhd (\alpha \wedge r(\overrightarrow{V})))))) \\
& && \{\text{Def of } (\lhd wait \rhd)\} \\
\Rightarrow & (II \lhd wait \rhd ((ok \wedge r(\overleftarrow{V}) \Rightarrow (ok' \wedge (\alpha \lhd wait' \rhd (\alpha \wedge r(\overrightarrow{V}))))))) && \{\text{Def of } \vdash\} \\
\Rightarrow & (r(\overleftarrow{V}) \vdash_{\mathcal{H}} (\alpha \lhd wait' \rhd \alpha \wedge r(\overrightarrow{V}))) && \square
\end{aligned}
$$

5.3. A Simple Example

We consider the following toy example, named CRUISE for convenience in exposition. In the system, the speed of the car V is continuous and is controlled by a computer by periodically calculating the acceleration a which must be applied for the next time unit. The sampling of the current speed is represented by the statement $v := V$. If $v < 80$, the car will accelerate and the rate of acceleration is computed by a control algorithm A. If $v \geq 80$, the car will decelerate at a rate determined by a control algorithm D. When accelerating, the behaviour of the car in the next time unit is defined by the phase statement $\ll \exists x.x = \overleftarrow{a} \wedge [\![|\dot{V} - x| \leq 3 \wedge \dot{V} \geq 1]\!] \wedge (\ell < 1 \lhd wait' \rhd \ell = 1) \gg$. Due to hardware limitations, the actual acceleration $\dot{V}$ may not be exactly the same as specified. Condition $|\dot{V} - x| \leq 3 \wedge \dot{V} \geq 1$ states that $\dot{V}$ is within an error of 3 from the value of x and that it is at least 1. The design task is to construct components A and D such that the speed never exceeds 100.

The requirement can be expressed by the following correctness formula

$$(\![V < 100]\!] \vee \ell = 0) : \ \{\mathbf{true}\} \quad \mathsf{CRUISE} \quad \{\mathbf{true}\}$$

Due to compositionality of our method, we can verify the correctness of the system based on the specifications of A and D. We record the verification by what is commonly called proof outline in program verification. In the outline, the formulae, marked by shaded boxes, represent conditions and conclusions of various verification rules used. For example, the formula in line (2) is obtained by applying the assignment axiom to the statement in line (1). The formula in turn serves as the pre-condition for the statement in line (3). Formula in line (6) is carried over from line (4), and is used as the pre-condition of the iteration

body; line (17) denotes the property that the body has, and by the iteration rule, the iteration statement satisfies the property in line (19).

$V := 0;$ (1)
$\{V = 0\}$ (2)
alarm := off; (3)
$\{V < 100\}$ (4)
while **true** do (5)
$\{V < 100\}$ (6)
$v := V;$ (7)
$\{V < 100 \wedge v = V\}$ (8)
if $v < 80$ (9)
then $\{V < 80\}$ A; $\{V < 80 \wedge 0 < a \leq 15\}$ (10)
$\ll \exists x.x = \overleftarrow{a} \wedge \|\dot{V} - x| \leq 3 \wedge \dot{V} \geq 1\| \wedge (\ell < 1 \lhd wait' \rhd \ell = 1) \gg$ (11)
$(\|V < 100\| \vee \ell = 0) \lhd wait' \rhd (\|V < 100\| \wedge l > 0.5 \wedge \overrightarrow{V} < 100)$ (12)
else $\{V < 100\}$ D; $\{V < 100\}$ (13)
$\ll \exists x.x = \overleftarrow{a} \wedge \|\dot{V} - x| \leq 3 \wedge \dot{V} \leq -1\| \wedge (\ell < 1 \lhd wait' \rhd \ell = 1) \gg$ (14)
$(\|V < 100\| \vee \ell = 0) \lhd wait' \rhd (\|V < 100\| \wedge l > 0.5 \wedge \overrightarrow{V} < 100)$ (15)
fi (16)
$(\|V < 100\| \vee \ell = 0) \lhd wait' \rhd (\|V < 100\| \wedge l > 0.5 \wedge \overrightarrow{V} < 100)$ (17)
od (18)
$(\|V < 100\| \vee \ell = 0)$ (19)

The proof outline indicates that to guarantee the simple invariant requirement, it is sufficient that algorithm A does not assign a value bigger than 15 to a, or formally

$$\{V < 80\} \;\; A \;\; \{V < 80 \wedge 0 < a \leq 15\}$$

It is preferable to accelerate fast when the speed is low and accelerate slowly when the speed is high. Therefore, we calculate the value of a based on function $0.1 * e^{100/(v+20)}$. To this end, we let

$$\{\mathbf{true}\} \;\; A \;\; \{a > 0 \wedge a \geq 0.1 * e^{100/(v+20)} - 1 \wedge a \leq 0.1 * e^{100/(v+20)}\}$$

Correctness of the system is maintained, because

$$0.1 * e^{100/(v+20)} \leq 15 \qquad\qquad \text{for any } v \geq 0$$

Finally, the control algorithm A is developed. It can be verified using Hoare Logic.

$c := 100/(v + 20);$
$k := 1;$
$a := 0.1;$
$r := 0.6 * 5;$
$\{a = 0.1 * \sum_{n=0}^{k-1} \frac{1}{n!}(\frac{100}{v+20})^n \wedge r = 0.6 * 5^k/k!\}$
while $k \leq 5 \vee r \geq 1$ do
  $a := a + 0.1 * c;$
  $k := k + 1;$
  $c := (100 * c)/((v + 20) * k);$
  $r := (5 * r)/k$
od
$\{a = 0.1 * \sum_{n=0}^{k-1} \frac{1}{n!}(\frac{100}{v+20})^n \wedge r = 0.6 * 5^k/k! \wedge k > 6 \wedge r < 1\}$
$\{a > 0 \wedge a \geq 0.1 * e^{100/(v+20)} - r \wedge a \leq 0.1 * e^{100/(v+20)}\}$

6. Conclusion

This paper adds a number of advanced features into DC, and aims to provide a semantical framework for sequential hybrid programs. In particular,

(1) A high order hiding operator is introduced to model local variables in real-time programs.

(2) A relational composition is formed to model the sequential composition operator of programming languages.

(3) Neighbourhood values of state variable are used to represent the stable states of program variables.

We have shown that the super-dense chop is the product of the ITL chop and the relational composition operator, where the former models the sequential composition of continuously evolving physical environment, and the latter composes discrete computing agents.

This paper also presents a link between the theory of designs and DC. It enables us to discuss time-independent properties of real-time programs in untimed refinement calculus, and as a result greatly simplifies the reasoning task for hybrid systems.

In this paper, the advanced features of DC are employed in formalising SHP, and provide verification methods for reasoning about timing properties. The same notations have been used to tackle concurrent real-time systems and provide a denotational semantics for an industrial specification language [LiH99].

Acknowledgements: we thank our colleagues for discussions and in particular Richard Moore for comments.

References

[Dij76] Dijkstra, E.W.: *A Discipline of Programming.* Prentice Hall, Englewood Cliffs, 1976.

[Hal83] Hale, R.W.S.: Programming in temporal logic. Technical report 42, Computing Laboratory, University of Cambridge, 1983.

[HCH93] Hale, R.W.S., Cardell-Oliver, R. and Herbert, J.M.J.: An embedding of timed transition system in HOL. *Formal Methods in System Design*, 3, 151–174 (1993).

[HMM83] Halpern, J., Manna, Z. and Moszkowski, B.: A hardware semantics based on temporal intervals. *Proc. of 10th International Colloquium on Automata, Languages and Programming*, 278–291, 1983.

[He93] He, J.F.: From CSP to hybrid systems. In A.W. Roscoe (ed): *A Classical Mind*, 171–190, Prentice Hall, 1993.

[HoH98] Hoare, C.A.R. and He, J.F.: *Unifying Theories of Programming.* Prentice Hall, 1998.

[Jon90] Jones, C.B.: *Systematic Software Development using VDM.* Prentice Hall, 1990.

[Koh88] Kohn, W.: A declarative theory for rational controllers. *Proc. 27th CDC*, 130–136, 1988.

[LiH99] Li, L. and He, J.F.: Timed RAISE and its application. *Proc. of the Sixth International Conference on Real-Time Computing Systems and Applications*, Hong Kong. IEEE Press, 1999.

[MMP92] Maler, O., Manna, Z. and Pnueli. A.: From timed to hybrid systems. *Proc. Real-Time: Theory in Practice*, LNCS 600, Springer-Verlag, 1992.

[Mat95] Mathematics of Construction Group: Fixed-point calculus. *Information Processing Letters*, 53: 131–136, (1995).

[Mor94] Morgan, C.C: *Programming from Specifications (2nd edition).* Prentice Hall, 1994.

[Mos85] Moszkowski, B.C.: A temporal logic for multi-level reasoning about hardware. *IEEE Computer*, 18(2), 10–19, 1985.

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

[Nic92] Nicolin, X.: An approach to the description and analysis of hybrid systems. LNCS 736, 121–149, Springer-Verlag, 1992.

[PaR95] Pandya, P. and Ramakrishna, Y.: A recursive Duration Calculus. Technical report, CS-95/3, Computer Science Group, TIFR, Bombay, India, 1995.

[PaD88] Pandya, P. and Dang, V.H.: Duration Calculus with weakly monotonic time. *Proc. of FTRTFT'98*, LNCS 1486, Springer-Verlag, 1998.

[PWX98] Pandya, P., Wang, H.P. and Xu, Q.W.: Towards a theory of sequential hybrid programs. *Proc. of IFIP Procomet Conference, Chapman & Hall*, 1998.

[Tar55] Tarski, A.: A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5, 285–309, 1955.

[Xu97] Xu, Q.W.: Semantics and verification of the extended phase transition systems in the Duration Calculus. *Proc. of International Workshop on Hybrid and Real-Time Systems*, LNCS 1201, 301–315, Springer-Verlag, 1997.

[ZHR91] Zhou, C.C., Hoare, C.A.R. and Ravn, A.P.: A Calculus of Duration. *Information Processing Letters*, 40(5): 269–275, (1991).

[ZRH93] Zhou, C.C., Ravn, A.P. and Hansen, M.: An extended Duration Calculus for hybrid systems. LNCS 736, 36–59, Springer-Verlag, 1993.

[ZH96] Zhou, C.C. and Hansen, M.: Chopping a point. *Proc. of BCS FACS 7th Refinement Workshop, Electronic Workshop in Computer Sciences*, Springer-Verlag, 1996.