A Denotational Semantics of Timed RSL Using Duration Calculus*

LI Li[2], HE Ji-feng[1,3]

[1](*International Institute for Software Technology, United Nations University, Macau, China*);

[2](*University of Science and Technology of China, Hefei* 230026, *China*);

[3](*East China Normal University, Shanghai* 200030, *China*)

E-mail: {ll,jifeng}@iist.unu.edu

http://www.iist.unu.edu/

Received July 12, 1999; accepted March 15, 2000

Abstract: This paper provides a denotational semantics to a subset of Timed RAISE Specification Language (RSL) using Extended Duration Calculus (EDC) model. It adds some novel features into the EDC model and explore their algebraic laws which play the vital role in formalising real-time programs and verification of real-time properties. Some algebraic laws of Timed RSL are presented, which can be proved from the denotational semantics, and be used in program transformation and optimization.

Key words: duration calculus; RAISE specification language; denotational semantics; real-time system

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 a formal specification of the components of the system is derived. 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 the delivered system requires that all the conceptual gaps between specification and implementation be closed.

For hybrid system a variety of formal methods have been developed, such as Phase Transition System[1,2], Declarative Control[3], the Extended State-Transition Graph[4] and the Hybrid CSP[5]. 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[6,7]. Moreover, state transitions of programs can also be characterised by properties relating the initial and final values of variables over interval of time[8,9]. But in the treatment of hybrid systems where the physical world evolves continuously, this approach seems inappropriate. Furthermore, we lose some important algebraic laws of the guarded command language in that framework, such as

* Supported by the National Natural Science Foundation of China under Grant No. 69773025 (国家自然科学基金)

LI Li was born in 1972. He is a Ph.D. candidate at the Department of Computer Science and Technology, University of Science and Technology of China. His research interests are formal methods, real-time system and multimedia communication. **HE Ji-feng** was born in 1943. He is a professor and doctorial supervisor of the East China Normal University and Shanghai Jiao Tong University. Currently he is working as a senior reserach fellow at the International Institute for Software Technology of the United Nations University. His research interests include software design methods, theories of programming and software/hardware co-design.

induction rules for recursions and the combination of assignments.

The RAISE Specification Language (RSL)[19] is one of the most versatile languages for formal specification, design and development of software systems. However, it has no particular features for real-time applications. This paper proposes a conservative extension of RSL for hybrid systems, based on the model of the Extended Duration Calculus (EDC)[11]. Our research is inspired by the pioneering work presented in Refs. [11~17]. 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 mathematical framework for Timed RSL equipped with delay and time-out constructs, proposed in Ref. [18];

3. Some algebraic laws of Timed RSL which play an important role in RAISE method, especially in justification.

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 rest of the paper is organised as follows. Section 1 gives a brief account of EDC and revisits its main constituents. We introduce some advanced features and explore their algebraic properties in Section 2. Section 3 provides a specification-oriented semantics for Timed RSL, and presents some algebraic laws. The paper ends with a brief summary and discussion.

1 Preliminaries

Like in EDC, we adopt continuous time represented by reals

$$Time =_{df} Real$$

An interval is represented by $[t1, t2]$ where $t1, t2 \in Time$ and $t1 \leqslant t2$. We use $Intv$ 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 catenation operator $\frown$

$$\sigma_1 \frown \sigma_2 =_{df} \sigma_1 \cup \sigma_2 \quad \textbf{if } \sigma_1.e = \sigma_2.b$$

Our formalism includes the following basic symbols, each associated with a type $Type(.)$.

- Global variables represent constant (i.e., independent of time) and are denoted by lower case letters $x, y, \ldots, z$.
- State variables represent functions on $Time$ and are denoted by capital letters $U, V, \ldots, W$,
- Temporal variables are identified as functions on intervals, and denoted by lower case letters $u, v, \ldots, w$.
- We use $f, g, \ldots, h$ for function names.
- $p, q, \ldots, r$ represent predicates.

A model $\mathcal{M}$ gives the meaning of every symbol:

- Global variable x is assigned a value $\mathcal{M}(x): Type(x)$.
- State variable V is interpreted as a time function $\mathcal{M}(V): Time \rightarrow Type(V)$.
- Temporal variable v is associated with an interval function

  $\mathcal{M}(v): Intv \rightarrow Type(v)$.

  A specific temporal variable l is used to denote the length of interval

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

- $\mathcal{M}$ assigns an n-ary function name f a function

$\mathcal{M}(f):Type_1\times\ldots\times Type_n\rightarrow Type(f)$. Constants are 0-ary function names interpreted as their value.

- An n-ary predicate name p is a special function name with type *Bool*. Boolean constants **true** and **false** are 0-ary predicate names interpreted as values *true* and *false*, respectively.

Let h be a variable. Two models $\mathcal{M}1$ and $\mathcal{M}2$ are called to be *h-equivalent*, denoted by $\mathcal{M}1\equiv_h\mathcal{M}2$, if for all variables v different from h

$$\mathcal{M}1(v)=\mathcal{M}2(v)$$

The state expressions of the language are defined by induction, and interpreted as functions on *Time*.

- Global and state variables are state expressions.

- If $E_1,\ldots,E_n$ are state expressions and f is an n-ary function name, then $f(E_1,\ldots,E_n)$ is also a state expression

$$\mathcal{M}(f(E_1,\ldots,E_n))(t)=_{df}\mathcal{M}(f)(\mathcal{M}(E_1)(t),\ldots,\mathcal{M}(E_n)(t))$$

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

- Global and temporal variables are terms.

- If $\theta_1,\ldots,\theta_n$ are terms and f is an n-ary function name, then $f(\theta_1,\ldots,\theta_n)$ is also a term

$$\mathcal{M}(f(\theta_1,\ldots,\theta_n))(\sigma)=_{df}\mathcal{M}(f)(\mathcal{M}(\theta_1)(\sigma),\ldots,\mathcal{M}(\theta_n)(\sigma))$$

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:

- Boolean typed terms are well-formed formulae.

- If $\theta_1,\ldots,\theta_n$ are terms, and p is an n-ary predicate name, then $p(\theta_1,\ldots,\theta_n)$ is a well-formed formula

$$\mathcal{M}(p(\theta_1,\ldots,\theta_n))(\sigma)=_{df}\mathcal{M}(p)(\mathcal{M}(\theta_1)(\sigma),\ldots,\mathcal{M}(\theta_n)(\sigma))$$

- If F and G are well-formed formulae, so are $\neg F$ and $F\wedge G$, $F\frown G$ and $\exists\ x\cdot F$, where x is a global variable.

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

$$\mathcal{M}(\exists\ x\cdot F)(\sigma)=_{df}\exists\ \mathcal{M}'\cdot\mathcal{M}'(F)(\sigma)\wedge(\mathcal{M}\equiv_x\mathcal{M}')$$

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

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

$$F\Rightarrow G=_{df}\neg F\vee G$$

$$\forall\ x\cdot F=_{df}\neg(\exists\ x\cdot\neg F)fg$$

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

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

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

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

Let b be a Boolean typed term. Define

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

In order to avoid an excessive number of parentheses the following precedence rules are used:

first: real operators

second: real predicates

third: $\neg,\Diamond,\Box$

fourth: $\frown$

fifth: $\wedge$

sixth: $<b\triangleright$

seventh: $\vee$

eighth: $\Rightarrow$

ninth: $\forall \cdot, \exists \cdot$

2 Advanced Features

2.1 Greatest lower bound and least upper bound

Definition 2.1 (Greatest lower bound).

Let $\mathscr{F}$ be a set of formulae. We define its greatest lower bound $\sqcap\mathscr{F}$ by

$$\mathscr{M}(\sqcap\mathscr{F})([t1,t2])=_{df}\exists\ F\in\mathscr{F}\cdot\mathscr{M}(F)([t1,t2])$$

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

(glb-1) $F\Leftarrow\sqcap\mathscr{F}$ **iff** $F\Leftarrow X$ for all $X\in\mathscr{F}$.

The greatest lower bound operator distributes over $\frown$.

Definition 2.2 (Least upper bound).

Let $\mathscr{F}$ be a set of formulae. We define its least upper bound $\sqcup\mathscr{F}$ by

$$\mathscr{M}(\sqcup\mathscr{F})([t1,t2])=_{df}\forall\ F\in\mathscr{F}\cdot\mathscr{M}(F)([t1,t2])$$

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

(lub-1) $F\Rightarrow\sqcup\mathscr{F}$ **iff** $F\Rightarrow X$ for all $X\in\mathscr{F}$.

2.2 Initial and final values

Definition 2.3 (Initial and final values).

Let V be a state variable. We introduce two terms $\mathbf{b}.V$ and $\mathbf{e}.V$ defined by

$$\mathscr{M}(\mathbf{b}.V)([t1,t2])=_{df}\mathscr{M}(V)(t1)$$

$$\mathscr{M}(\mathbf{e}.V)([t1,t2])=_{df}\mathscr{M}(V)(t2)$$

For a state expression E, its initial value $\mathbf{b}.E$ and final value $\mathbf{e}.E$ can be defined in the same way.

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

(V-1) (intermediate value) $(F\wedge p(\mathbf{e}.V))\frown G=F\frown(p(\mathbf{b}.V)\wedge G)$

(V-2) (initial value) $(p(\mathbf{b}.V)\wedge F)\frown G=p(\mathbf{b}.V)\wedge(F\frown G)$

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

(V-4) (point value) $(l=0)\wedge p(\mathbf{e}.V)=p(\mathbf{b}.V)\wedge(l=0)$

2.3 Stability

Definition 2.4 (Stability)

Let S be a Boolean state expression. Like in EDC, a Boolean typed term $\lceil S\rceil$ is defined by

$$\mathscr{M}(\lceil S\rceil)([t1,t2])=_{df}t1<t2\wedge\forall\ t\in(t1,t2)\cdot\mathscr{M}(S)(t)$$

The following law indicates that $\lceil\cdot\rceil$ is a derived operator.

($stb-0$) (stability and initial value)

$$\lceil S\rceil=\neg((l>0)\frown\neg\mathbf{b}.S\frown(l>0))\wedge l>0$$

$\lceil\cdot\rceil$ enjoys the following algebraic properties.

(stb-1) $\lceil\mathbf{true}\rceil=(l>0)$

(stb-2) $\lceil\mathbf{false}\rceil=\mathbf{false}$

(stb-3) $\lceil S_1\wedge S_2\rceil=\lceil S_1\rceil\wedge\lceil S_2\rceil$

We define

$$\lfloor S\rfloor=_{df}\neg S\neg\vee l=0 \quad \|S\|=_{df}\mathbf{b}.S\wedge\lfloor S\rfloor\wedge\mathbf{e}.S$$
$$\lceil S\rceil=_{df}\lceil S\rceil\wedge\mathbf{e}.S \quad \lfloor S\rfloor=_{df}\lceil S\rceil\vee l=0$$

2.4 Left and right limits

Definition 2.5 (Left and righ: limits).

A state variable V is *finitary* if for all models $\mathscr{M}$ and all time intervals σ, σ can be divided into finite number of sub-intervals, so that $\mathscr{M}(V)$ keeps constant in each sub-interval.

We introduce the left limit $\overleftarrow{V}$ and right limit $\overrightarrow{V}$ of a finitary state variable V as terms defined by

$$\mathscr{M}(\overleftarrow{V})([t1,t2])=c \text{ if } \exists\ \delta>0\cdot\mathscr{M}(\lfloor V=c\rfloor)([t1-\delta,t1])=true$$
$$\mathscr{M}(\overrightarrow{V})([t1,t2])=c \text{ if } \exists\ \delta>0\cdot\mathscr{M}(\lfloor V=c\rfloor)([t2,t2+\delta])=true$$

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

The concepts of limit are captured by the following laws:

(limit-1) (removal of limit)

$$(F\wedge p(\overrightarrow{V}))\frown(G\wedge(l>0))=F\frown(G\wedge\exists\ x\cdot(p(x)\wedge(\lceil V=x\rceil\frown\mathbf{true})))$$
$$(F\wedge(l>0))\frown(p(\overleftarrow{V})\wedge G)=(F\wedge\exists\ x\cdot(p(x)\wedge(\mathbf{true}\frown\lceil V=x\rceil)))\frown G$$

(limit-2) (limit of composition)

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

We define $\overleftarrow{\ }S\|=_{df}\overleftarrow{S}\wedge\lfloor S\|,\lfloor S\|=_{df}\lfloor S\|\wedge\overrightarrow{S}$, $\overrightarrow{\lfloor S\|}=_{df}\overrightarrow{S}\wedge\lfloor S\|\wedge\overrightarrow{S}$.

2.5 Hiding state variable

Definition 2.6 (Hiding state variable).

Let V be a state variable. Define

$$\mathscr{M}(\exists\ V\cdot F)([t1,t2])=_{df}\exists\ \mathscr{M}'\cdot\mathscr{M}'(F)([t1,t2])\wedge(\mathscr{M}=_V\mathscr{M}')$$

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

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

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

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

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

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

($\exists$-4) (hiding and substitution)

$$F=\exists\ W\cdot(F[W/V]\wedge\|W=V\|\wedge(\overleftarrow{W}=\overleftarrow{V})\wedge(\overrightarrow{W}=\overrightarrow{V}))$$

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

$$F=\exists\ W\cdot(F[W/V]\wedge\|W=V\|\wedge(\overleftarrow{W}=\overleftarrow{V}))$$
$$G=\exists\ W\cdot(G[W/V]\wedge\|W=V\|\wedge(\overrightarrow{W}=\overrightarrow{V}))$$

2.6 Chopping points

Definition 2.7 (Superdense chop).

Let F and G be formulae of state variable V. Define

$$F\cdot G=_{df}\exists\ x,V_F,V_G\cdot(F[V_F/V]\wedge\|V=V_F\|\wedge(\overrightarrow{V}=\overrightarrow{V_F})\wedge(V_F^{\cdot}=x))\frown$$
$$(G[V_G/V]\wedge\|V=V_G\|\wedge(\overleftarrow{V}=\overleftarrow{V_G})\wedge(\overleftarrow{V_G}=x))$$

The chop operator $\frown$ is used to compose the continuously evolving hybrid systems, whereas the relational

composition operator is used to model the sequential composition of imperative programming languages. The following theorem states that $\circ$ can be seen as the product of the chop operator $\frown$ and the relational composition operator,

Theorem 2.8. If $\overleftarrow{V}$ and $\vec{V}$ do not occur in F nor G, then

$$(F \wedge p(\overleftarrow{V},\vec{V})) \circ (q(\overleftarrow{V},\vec{V}) \wedge G) = (F \frown G) \wedge \exists\, x \cdot (p(\overleftarrow{V},x) \wedge q(x,\vec{V}))$$

Proof.

LHS {($\exists$ -2) and Def. of $\circ$}

$=\exists\, x \cdot (\exists\, V_1 \cdot (F[V_1/V] \wedge [\![V=V_1]\!]) \wedge \exists\, V_1 \cdot ((\overleftarrow{V}=\overleftarrow{V_1}) \wedge (\vec{V_1}=x) \wedge p(\overleftarrow{V_1},\vec{V_1})))$

$\quad \frown (\exists\, V_2 \cdot (G[V_2/V] \wedge [\![V-V_2]\!]) \wedge \exists\, V_2 \cdot ((\overleftarrow{V}=\overleftarrow{V_2}) \wedge (\overleftarrow{V_2}=x) \wedge q(\overleftarrow{V_2},\overleftarrow{V_2})))$

Corollary of ($\exists$ -4)}

$=\exists\, x \cdot ((F \wedge p(\overleftarrow{V},x)) \frown (G \wedge q(x,\vec{V})))$ {limit-2)}

$=RHS$ □

Theorem 2.9 ($\circ$ and $\frown$)

If $\vec{V} \notin F$ and $\overleftarrow{V} \notin G$, then $F \circ G = F \frown G$.

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

Theorem 2.10.

(1) (associativity) $(F \circ G) \circ H = F \circ (G \circ H)$

(2) (unit) $F \circ I = F = I \circ F$, where $I =_{df} (l=0) \wedge (\vec{V}=\overleftarrow{V})$

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

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

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

(5) (initial stable state) $(p(\overleftarrow{V}) \wedge F) \circ G = p(\overleftarrow{V}) \wedge (F \circ G)$

(6) (final stable state) $F \circ (G \wedge q(\vec{V})) = (F \circ G) \wedge q(\vec{V})$

(7) (consistency) $(F \wedge r(\vec{V})) \circ G = F \circ (r(\overleftarrow{V}) \wedge G)$

(8) (invisible stable state) *If* $\vec{V} \notin F$ and $\overleftarrow{V} \notin G$, then

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

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

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

2.7 Weakest fixed points

Definition 2.11 (Weakest fixed point)

Let Φ be a monotonic mapping of formulae. We define its weakest fixed point by

$$\mu X \cdot \Phi(X) =_{df} \sqcap \{F \mid F \Rightarrow \Phi(F)\}$$

From Tarski's Fixed Point Theorem[19] it follows that $\mu X \cdot \Phi(X)$ is subject to the following laws.

(μ-1) (fixed point) $\Phi(\mu X \cdot \Phi(X)) = \mu X \cdot \Phi(X)$.

(μ-2) (weakest fixed point) If $F \rightarrow \Phi(F)$, then $F \rightarrow \mu X \cdot \Phi(X)$.

3 Denotational Semantics of Timed RSL

In this paper, we consider a subset of Timed RSL expressions, which has the form:

chaos | **stop** | **skip** | r | v := r | P ; Q | **let** x = P **in** Q **end** | cc | wait r |

P □ Q | **if** r **then** P **else** Q **end** | $\square_{i \in I}$ G_i | **while** r **do** P **end** | P ‖ Q | P ⫴ Q

where r ranges over read-only expressions, i.e. expressions not involving assignment, input, output or **wait**, v over variables, P and Q over expressions, x over bindings (a binding is a structure of identifiers, possibly grouped

by parentheses), cc over inputs and outputs of the form c? or c!r. C_i has the form C_i, M_i, where C_i ranges over expressions of the form: **let** $x_i = cc_i$ **in** P_i **end**, M_i over expressions of the form **wait** $r_i; Q_i$. $\prod_{i\in\{}G_i$ is defined as **stop**.

In RSL, sequencing (;) is a derived notion, and it is defined in terms of **let**. We include it to give a conventional presentation.

The semantics of a Timed RSL expression is observations on its channels, and observations on its variables and return value when it terminates. We will use the following observables:

Tr: a state variable representing the sequence of communications which have been recorded so far, also called *trace*. We use $Tr \downarrow A$ to denote the subsequence of Tr of communications on channels in set A.

V_i: a state variable recording the current value of Timed RSL variable v_i.

V: the binding of all state variables V_i: $(V_1, \ldots, V_n)$.

u: the binding of global variables corresponding to V: $(u_1, \ldots, u_n)$.

$r(\vec{V})$: a term constructed by replacing each occurrence of Timed RSL variable v_i in the read-only Timed RSL expression r with temporal variable $\vec{V}_i$, each occurrence of Timed RSL identifier x_i with global variable x_i. It is used to map a read-only Timed RSL expression to an EDC term, using the current value of all Timed RSL variables.

W: a state variable recording the value returned by a terminating expression.

Ref: a state variable recording the current set of communications which may be refused by the expression. An element $c_i?(c_i!)$ in the set represents that the expression may refused input from (output to) channel c_i.

ok: a global Boolean variable recording the observation that the expression has been started.

ok': a global Boolean variable recording the observation that the expression has finished successfully.

$wait$: a global Boolean variable, which is true when the expression is asked to start in a waiting state of its predecessor.

$wait'$: a global Boolean variable, which is true when the expression is in a state waiting for communication with the environment.

We use an EDC formula to define the semantics of a Timed RSL expression. Since the execution of a Timed RSL expression can never undo any communication performed previously, the trace can only get longer. A formula P which defines the semantics of a Timed RSL expression must therefore imply this fact. So it satisfies the healthiness condition:

$(H1)$ $P = P \wedge R1$ where

$$R1 =_{df} \exists\, s \cdot (\vec{Tr} = s \wedge \Box(s \leqslant \mathbf{b}.Tr \wedge \mathbf{b}.Tr \leqslant \mathbf{e}.Tr) \wedge (l = 0 \vee \mathbf{e}.Tr \leqslant \vec{Tr}) \wedge \vec{Tr} \leqslant \vec{Tr})$$

where $\Box F =_{df} \neg(l > 0) \frown \neg F \frown \mathbf{true}$.

Theorem 3.1. $R1 \circ R1 = R1$.

Definition 3.2 (Sequential composition).

$$P;Q =_{df} \exists\, w, o \cdot (P^{-}_{-}[w, o/wait', ok']^{-} \cdot Q[w, o/wait, ok]).$$

The main purpose of global variable $wait'$ is to distinguish intermediate observations from the observations made on termination. In sequential composition, the intermediate observations 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:

$(H2)$ $P = II \lhd wait \rhd P$ where $II =_{df}$

$$(\mathbf{true} \vdash wait' = wait \wedge \vec{Tr} = \vec{Tr} \wedge \vec{V} = \vec{V} \wedge \vec{Ref} = \vec{Ref} \wedge \vec{W} = \vec{W} \wedge l = 0) \wedge R1$$

and $P \vdash Q =_{df} ok \wedge P \Rightarrow ok' \wedge Q$.

Definition 3.3 (Healthy formula). P is healthy if it satisfies $H1$ and $H2$.

Theorem 3.4. P is healthy iff it satisfies $P=\mathscr{H}(P)$ where

$$\mathscr{H}(X)=_{df} II \lhd wait \rhd (X \wedge R1)$$

Theorem 3.5. For any EDC formula P, $\mathscr{H}(P)$ is a healthy formula.

A Timed RSL expression must conform to the basic healthiness condition of sequential processes, that it makes no prediction about a process that has not started (except that the trace can only get longer); and it is monotonic with the variable ok':

$(H3)$ $P=P \vee \neg ok \wedge R1$

$(H4)$ $P\mathbf{false}/ok' \Rightarrow P\mathbf{true}/\mathrm{ok}'$

Definition 3.6 (Healthy design). A healthy design is an EDC formula having the form: $\mathscr{H}(P \vdash W \lhd wait' \rhd T)$. $\mathscr{D}$ will stand for the set of healthy designs.

Theorem 3.7. Healthy design satisfies $H3$ and $H4$.

If a Timed RSL expression described by a healthy design $\mathscr{H}(P \vdash W < wait' \rhd T)$ starts and its behaviour is not divergence as described in $\neg P$, it will finish successfully and its waiting behaviour is described by W, its terminating behaviour is described by T. Therefore, we use the following definition of the semantics of a Timed RSL expression P:

Definition 3.8 (Semantics of Timed RSL expression)

$$[\![P]\!] =_{df} \mathscr{H}(\neg [\![P]\!]_{div} \vdash [\![P]\!]_{wait} \lhd \mathrm{wait}' [\![P]\!]_{ter})$$

where $[\![P]\!]_{div} \Rightarrow \mathrm{R1}$ and $[\![P]\!]_{wait} \Rightarrow \mathrm{R1}$ *and* $[\![P]\!]_{ter} \rightarrow \mathrm{R1}$.

Theorem 3.9 (Closure of glb and lub)

(1) $\sqcap_i \mathscr{H}(P_i \vdash W_i \lhd wait' \rhd T_i) = \mathscr{H}(\sqcup_i P_i \vdash \sqcap_i W_i < wait' \rhd \sqcap_i T_i)$

(2) $\sqcup_i \mathscr{H}(P_i \vdash W_i \lhd wait' \rhd T_i) = \mathscr{H}(\sqcap_i P_i \vdash \sqcup_i (P_i \Rightarrow W_i) < wait' \rhd \sqcup_i (P_i \Rightarrow T_i))$

Theorem 3.10. Healthy design forms a complete lattice.

The weakest fixed point of a monotonic mapping of healthy designs can therefore be defined by: $\mu_{\mathscr{D}} X \cdot \Phi(X) =_{df} \sqcap \{D \mid D \Rightarrow \Phi(D), D \in \mathscr{D}\}$.

Theorem 3.11 (Refinement ordering)

$[\![P]\!] \Rightarrow [\![Q]\!]$ iff

$$([\![P]\!]_{div} \rightarrow [\![Q]\!]_{div}) \wedge ([\![P]\!]_{wait} \rightarrow [\![Q]\!]_{div} \vee [\![Q]\!]_{wait}) \wedge ([\![P]\!]_{ter} \Rightarrow [\![Q]\!]_{div} \vee [\![Q]\!]_{ter})$$

Corollary 3.12.

$$[\![P]\!] = [\![Q]\!] \text{ iff } ([\![P]\!]_{div} = [\![Q]\!]_{div}) \wedge (([\![P]\!]_{div} \vee [\![P]\!]_{wait}) - ([\![Q]\!]_{div} \vee [\![Q]\!]_{wait})) \wedge (([\![P]\!]_{div} \vee [\![P]\!]_{ter}) = ([\![Q]\!]_{div} \vee [\![Q]\!]_{ter}))$$

Definition 3.13 (Equivalence). Two Timed RSL expressions P and Q are equivalent, denoted by $\mathrm{P} \simeq \mathrm{Q}$, which is defined by

$$\mathrm{P} \simeq \mathrm{Q} =_{df} [\![\mathrm{P}]\!] = [\![\mathrm{Q}]\!].$$

To define the semantics of Time RSL expressions, we use the following abbreviations.

An expression is *stability*, if either the length of interval is zero, or the trace remains unchanged in a right closed interval:

$$Stb^w =_{df} \exists\, s \cdot (\lfloor Tr = s \rfloor)$$

$$Stb' =_{df} \exists\, s \cdot (\lfloor Tr = s \rfloor)$$

Here Stb^w describes stability in waiting behaviour, while Stb' describes stability in terminating behaviour, where the right limit of trace may be changed by communication.

The *output* of an expression is the right limit of the trace, all state variables, and the return value when it

terminates:

$$O(\mu,\nu,w)=_{df}\overline{Tr}=\mu\wedge\vec{V}=\nu\wedge\vec{W}=w$$

Chaos. The expression **chaos** *is the worst expression, whose behaviour is totally unpredictable*:

$$[\![\ \textbf{chaos}\ ]\!]=_{df}\mathscr{H}(\textbf{false}\vdash\textbf{true}<wait'>\textbf{true})=\mathscr{H}(\textbf{true})$$

Stop. The expression **stop** never communicates with its environment. It does not diverge but stays in idle forever, for it does not pursue any internal computation nor terminate:

$$[\![\ \textbf{stop}\ ]\!]=_{df}\mathscr{H}(\textbf{true}\vdash Stb^{\omega}>wait'<\textbf{false})$$

Skip. The expression **skip** terminates immediately and returns the unit value () of type **Unit**:

$$[\![\ \textbf{skip}\ ]\!]=_{df}\mathscr{H}(\textbf{true}\vdash\textbf{false}<wait'>l=0\wedge O(\overline{Tr},\vec{V},()))$$

Read-only expression. The read-only expression terminates immediately and returns its value evaluated by values of the left limit of all variables:

$$[\![\ r\ ]\!]=_{df}\mathscr{H}(\textbf{true}\vdash\textbf{false}<wait'>l=0\wedge O(\overline{Tr},\vec{V},r(\vec{V})))$$

Assignment. The assignment expression $v_i:=r$ terminates immediately, changes the value of v_i to the value of r, and returns unit value () of type **Unit**:

$$[\![\ v_i:=r\ ]\!]=_{df}\mathscr{H}(\textbf{true}\vdash\textbf{false}<wait'>l=0\wedge O(\overline{Tr},\vec{V}[r(\vec{V})/\vec{V}_i],()))$$

Sequential composition. The sequential composition P; Q executes P first until it terminates and then runs Q:

$$[\![\ \textbf{P};\textbf{Q}\ ]\!]=_{df}[\![\ \textbf{P}\ ]\!];[\![\ \textbf{Q}\ ]\!]$$

where ; on the right hand side stands for the sequential composition operator defined in Definition 3.2.

Theorem 3.14 (Sequential composition)

$$[\![\ \textbf{P};\textbf{Q}\ ]\!]_{div}=([\![\ \textbf{P}\ ]\!]_{ter}\circ[\![\ \textbf{Q}\ ]\!]_{div})\vee[\![\ \textbf{P}\ ]\!]_{div}$$

$$[\![\ \textbf{P};\textbf{Q}\ ]\!]_{wait}=([\![\ \textbf{P}\ ]\!]_{ter}\circ[\![\ \textbf{Q}\ ]\!]_{wait})\vee[\![\ \textbf{P}\ ]\!]_{wait}$$

$$[\![\ \textbf{P};\textbf{Q}\ ]\!]_{ter}=([\![\ \textbf{P}\ ]\!]_{ter}\circ[\![\ \textbf{Q}\ ]\!]_{ter})$$

Local identifiers can be introduced by existential quantifier:

$$[\![\ \textbf{let}\ x=\textbf{P}\ \textbf{in}\ \textbf{Q}\ \textbf{end}\ ]\!]=_{df}[\![\ \textbf{P}\ ]\!];\exists\ x\cdot(x=\vec{W}\wedge[\![\ \textbf{Q}\ ]\!])$$

It can be proved that if x is not free in Q then P; Q~**let** x=P **in** Q **end**. This is the definition of sequencing (;) in RSL.

Theorem 3.15 (**chaos**, **stop**, **skip**, assignment, and sequential composition)

(1) **chaos**; P≃**chaos**

(2) **stop**; P≃**stop**

(3) **skip**; P≃P

(4) $v:=e_1;v:=e_2\simeq v:=e_2[e_1/v]$

Proof of (1). We use "PL" to refer to Predicate Logic.

1. [**chaos**] {Def.}

$=\mathscr{H}(\textbf{false}\vdash\textbf{true}>wait'>\textbf{true})$ {PL}

$=\mathscr{H}(\neg R1\vdash R1<wait'>R1)$

2. $[\![\ \textbf{chaos}\ ]\!]_{div}=[\![\ \textbf{chaos}\ ]\!]_{wait}=[\![\ \textbf{chaos}\ ]\!]_{ter}=R1$ {1, Def. 3.8}

3. [**chaos**; P] {2, Th. 3.14}

$=\mathscr{H}(\neg((R1\circ[\![\ \textbf{P}\ ]\!]_{div})\vee R1)\vdash(R1\circ[\![\ \textbf{P}\ ]\!]_{wait})\vee R1<wait'>R1\circ[\![\ \textbf{P}\ ]\!]_{ter}$

{Def. 3.8, Th. 2.10(3), Th. 3.1}

$=\mathscr{H}(\neg R1\vdash R1<wait'>R1\circ[\![\ \textbf{P}\ ]\!]_{ter}$ {Coro. 3.12}

$=\mathscr{H}(\neg R1\vdash R1<wait'>R1\vee(R1\circ[\![\ \textbf{P}\ ]\!]_{ter}))$ {Def. 3.8, Th. 2.10(3), Th. 3.1}

$=\mathscr{H}(\neg R1 \vdash R1 \lhd wait' \rhd R1)$ {1.}

$=$ ⟦ **chaos** ⟧

Communication. The communication expression is stable while waiting for synchronization with its partner. As soon as its partner is ready, the communication will take place, the trace will be updated, and the time elapsed while waiting (and the value received if the expression is input) will be returned:

$$⟦ c?\ ⟧ =_{df} Active(\{c?\});\ ⟦ c?'\ ⟧$$

$$⟦ c!r\ ⟧ =_{df} Active(\{c!\});\ ⟦ c!r'\ ⟧$$

where

$$Active(C) =_{df} \mathscr{H}(\textbf{true} \vdash W(C) \lhd wait' \rhd T(C))$$

$$W(C) =_{df} Stb^{w} \wedge [C \cap Ref = \{\}]$$

$$T(C) =_{df} \exists\ \tau \cdot (Stb' \wedge [C \cap Ref = \{\}] \wedge l = \tau \wedge O(\vec{Tr}, \vec{V}, \tau))$$

$$⟦ c?'\ ⟧ =_{df} \mathscr{H}(\textbf{true} \vdash \textbf{false} \lhd wait' \rhd ⟦ c?'\ ⟧_{ter})$$

$$⟦ c?'\ ⟧_{ter} =_{df} \exists\ m \cdot (l = 0 \wedge O(\vec{Tr}^{\wedge}\langle c.m\rangle, \vec{V}, (m, \vec{W})))$$

$$⟦ c!r'\ ⟧ =_{df} \mathscr{H}(\textbf{true} \vdash \textbf{false} \lhd wait' \rhd ⟦ c!r'\ ⟧_{ter})$$

$$⟦ c!r'\ ⟧_{ter} =_{df} l = 0 \wedge O(\vec{Tr}^{\wedge}\langle c.r(\vec{V})\rangle, \vec{V}, \vec{W})$$

Here $Active(C)$ describes the stable behaviour while waiting for synchronization with its partner (not refusing any communication in the set C). When terminates, it returns the time elapsed while waiting. ⟦ cc' ⟧ describes the occurence of communication at a time point.

Theorem 3.16 (communication, assignment and sequential composition)

$$c?;v:=r;\ \textbf{skip} \simeq v:=r;\ c?;\ \textbf{skip}$$

Wait. The expression **wait** r remains idle for r time units, and then terminates with return value unit, leaving all program variables unchanged. Here r is a read-only expression of type **Time**, which is defined as the set of all non-negative real numbers:

$$⟦\ \textbf{wait}\ r\ ⟧ =_{df} Active(\{\})[r(\vec{V})];\ ⟦\ \textbf{skip}\ ⟧$$

where $Active(C)[\varepsilon] =_{df} \mathscr{H}(\textbf{true} \vdash W(C) \wedge l < \varepsilon \lhd wait' \rhd T(C) \wedge l = \varepsilon)$.

Theorem 3.17 (**wait** and sequential composition)

(1) **wait** $0 \simeq$ **skip**

(2) **wait** r_1; **wait** $r_2 \simeq$ **wait** $(r_1 + r_2)$

Proof of (1). We use "ITL" to refer to Interval Temporal Logic.

⟦ **wait** 0 ⟧ {Def.}

$=\mathscr{H}$**true** $\vdash W(\{\}) \wedge l < 0 \lhd wait' \rhd T(\{\}) \wedge l = 0;$ ⟦ **skip** ⟧ {ITL, Def.}

$=\mathscr{H}$(**true** $\vdash$ **false** $\lhd wait' \rhd O(\vec{Tr}, \vec{V}, 0) \wedge l = 0;$

$\mathscr{H}$**true** $\vdash$ **false** $\lhd wait' \rhd O(\vec{Tr}, \vec{V}, ()) \wedge l = 0$ {Th. 3.14, Th. 2.10(4)}

$=\mathscr{H}$**true** $\vdash$ **false** $\lhd wait' \rhd (O(\vec{Tr}, \vec{V}, 0) \wedge l = 0) \circ (O(\vec{Tr}, \vec{V}, ()) \wedge l = 0)$ {Th. 2.8, ITL}

$=\mathscr{H}$(**true** $\vdash$ **false** $\lhd wait' \rhd (O(\vec{Tr}, \vec{V}, ()) \wedge l = 0)$ {Def.}

$=$ ⟦ **skip** ⟧

Internal choice. The expression P⌈⌉Q behaves like either P or Q. The choice between them is made non-deterministically:

$$⟦\ P\ ⌈⌉\ Q\ ⟧ =_{df} [\ ⟦ P ⟧ \vee ⟦ Q ⟧\ ]$$

Conditional. The expression **if** r **then** P **else** Q **end** executes P if the initial value of r is true, otherwise it executes Q instead:

$$⟦\ \textbf{if}\ r\ \textbf{then}\ P\ \textbf{else}\ Q\ ⟧ =_{df} ⟦ P ⟧ \lhd r(\vec{V}) \rhd ⟦ Q ⟧$$

External choice. The expression $\Box_{i\in I}$ **let** $x_i=cc_i$ **in** P_i **end** is willing to do any communication described in the set $C=\{c_i?\mid cc_i \text{ is } c_i?r_i, i\in I\}\cup\{c_i!\mid cc_i \text{ is } c_i!r_i, i\in I\}$ until one of them (for example cc_i) occurs and executes the corresponding expression (P_i):

$$[\![\ \Box_{i\in I}\textbf{let } x_i=cc_i \textbf{ in } P_i \textbf{ end}\ ]\!]=_{df} Active(C);\ \forall_{i\in I}[\![\ \textbf{let } x_i=cc_i' \textbf{ in } P_i \textbf{ end}\ ]\!]$$

The expression $\Box_{i\in I}$ **let** $x_i=cc_i$ **in** P_i **end** $\Box\Box_{j\in J}$ **wait** $r_j;Q_j$ is willing to do any communication described in the set C defined above until one of them (for example cc_i) occurs and executes the corresponding expression (P_i) or the fastest **wait** expression (for example **wait** r_j) terminates and executes the corresponding expression (Q_j):

$$[\![\ \Box_{i\in I}\textbf{let } x_i=cc_i \textbf{ in } P_i \textbf{ end}\Box\Box_{j\in J}\textbf{wait } r_j;Q_j\ ]\!]=_{df}$$
$$\exists\, r\cdot(r=\mathrm{Min}_{j\in J}(r_j(\bar{V}))\wedge(Active(C)(r);\vee_{i\in I}[\![\ \textbf{let } x_i=cc_i' \textbf{ in } P_i \textbf{ end}\ ]\!])\vee$$
$$(Active(C)[r];\vee_{j\in J}([\![\ Q_j\ ]\!]\wedge r_j(\bar{V})=r)))$$

where $Active(C)(\varepsilon)=_{df}\mathscr{H}(\textbf{true}\vdash W(C)\wedge l<\varepsilon<wait'\rhd T(C)\wedge l\leqslant\varepsilon)$.

Theorem 3.18 (internal choice and external choice)

let $x_1=cc_1$ **in** P **end** $\Box$ **let** $x_1=cc_1$ **in** Q **end**$\simeq$

let $x_1=cc_1$ **in** P $\Box$Q **end**

Iteration. It can be easily verified that the conditional operator ($\lhd r(\bar{V})\rhd$) and the sequential operator ($;$) are monotonic, so we can define the semantics of iteration as the *weakest fixed point*:

$$[\![\ \textbf{while r do P end}\ ]\!]=_{df}\mu_{\mathscr{D}}X\cdot(([\![\ P\ ]\!];X)\lhd r(\bar{V})\rhd[\![\ \textbf{skip}\ ]\!])$$

Theorem 3.19 (**skip** and iteration)

while true do skip end$\simeq$**chaos**

Concurrent composition. In the expression P‖Q, P and Q are assignment disjoint, which means neither of them can write variables that can be read or written by the other. Both of them return unit. They are executed concurrently until one of them terminates, when the other continues. When a communication can take place, it will occur immediately. This is so called *maximal progress*. The communication between P and Q is not visible. We use $\alpha(P)$ to represent the set of variables P can access, and use $V_{\alpha(P)}$ instead of V:

$$\alpha(P\|Q)=_{df}\alpha(P)\cup\alpha(Q)$$
$$[\![\ P\|Q\ ]\!]_{div}=_{df}\exists\, Ref_0,Ref_1,Tr_0,Tr_1\cdot([\![\ P\ ]\!]^w_{div}[0]\wedge[\![\ Q\ ]\!]^w_{div}[1]\vee[\![\ P\ ]\!]_{div}[0]\wedge([\![\ Q\ ]\!]^w_{wait}[1]\vee[\![\ Q\ ]\!]^w_{ter}[1])\vee$$
$$[\![\ Q\ ]\!]_{div}[1]\wedge([\![\ P\ ]\!]^w_{wait}[0]\vee[\![\ P\ ]\!]^w_{ter}[0]))\cdot R1$$
$$[\![\ P\|Q\ ]\!]_{wait}=_{df}\exists\, Ref_0,Ref_1,Tr_0,Tr_1\cdot([\![\ P\ ]\!]^w_{wait}[0]\wedge[\![\ Q\ ]\!]^w_{wait}[1]\vee$$
$$[\![\ P\ ]\!]_{wait}[0]\wedge[\![\ Q\ ]\!]^w_{ter}[1]\vee[\![\ Q\ ]\!]_{wait}[1]\wedge[\![\ P\ ]\!]^w_{ter}[0])$$
$$[\![\ P\|Q\ ]\!]_{ter}=_{df}\exists\, Ref_0,Ref_1,Tr_0,Tr_1\cdot([\![\ P\ ]\!]_{ter}[0]\wedge[\![\ Q\ ]\!]_{ter}[1]\vee[\![\ Q\ ]\!]_{ter}[1]\wedge[\![\ P\ ]\!]^w_{ter}[0])$$

where

$$[\![\ P\ ]\!]^w_{div}[i]=_{df}[\![\ P\ ]\!]_{div}[i]\wedge Conc$$
$$[\![\ P\ ]\!]^w_{wait}[i]=_{df}[\![\ P\ ]\!]_{wait}[i]\wedge Conc$$
$$[\![\ P\ ]\!]^w_{ter}[i]=_{df}([\![\ P\ ]\!]_{ter}[i\cdot CIdle^w(i)])\wedge Conc$$
$$[\![\ P\ ]\!]^{\bar{w}}_{ter}[i]=_{df}([\![\ P\ ]\!]_{ter}[i\cdot CIdle'(i)])\wedge Conc$$
$$P[i]=_{df}P[Ref_i/Ref][Tr_i/Tr]$$
$$Conc=_{df}Maxp\wedge\exists\, s,s_0,s_1\cdot(Tr=s\wedge\bar{Tr}_0=s_0\wedge\bar{Tr}_1=s_1\wedge$$
$$\lfloor(Tr-s)\in(Tr_0-s_0)\|(Tr_1-s_1)\wedge Ref=Ref_0\cap Ref_1\rfloor)$$

$$Maxp =_{df} \bigvee \lfloor \neg (c_i? \notin Ref_0 \wedge c_i! \notin Ref_1 \vee c_i? \notin Ref_2 \wedge c_i! \notin Ref_0) \rfloor^*$$

$$\langle\rangle \| t =_{df} \{t\}$$

$$(\langle x \rangle^\frown s) \| (\langle y \rangle^\frown t) =_{df} \left\{ z^\frown u \;\middle|\; \begin{array}{l} z = \langle x \rangle \wedge u \in (s \| (\langle y \rangle^\frown t)) \vee \\ z = \langle y \rangle \wedge u \in ((\langle x \rangle^\frown s) \| t) \vee \\ z = \langle\rangle \wedge x = y \wedge u \in (s \| t) \end{array} \right\}$$

$$CIdle^w(i) =_{df} \exists\, s_i (\lfloor Tr_i = s_i \rfloor^*)$$

$$CIdle^t(i) =_{df} CIdle^w(i) \wedge \overleftarrow{V}_{a_j} = \overrightarrow{V}_{a_j}$$

$$a_1 =_{df} \alpha(P) \qquad a_2 =_{df} \alpha(Q)$$

Here *Maxp* represents the phenomenon *maximal progress*, *Conc* describes the concurrent execution, the last line of $\|$ definition between traces in the braces hides the communication between P and Q.

In RSL, channels can be hided by **local** declaration, so that they are not visible outside the **local** expression. We use P\A to represent the expression P which has the set A of channels hided by **local** declaration:

$$[\![P\backslash A]\!]_{dfo} =_{df} [\![P]\!]_{dfo} \backslash A \quad [\![P\backslash A]\!]_{wait} =_{df} [\![P]\!]_{wait} \backslash A \quad [\![P\backslash A]\!]_{ter} =_{df} [\![P]\!]_{ter} \backslash A$$

where $P\backslash A =_{df} (P \wedge \exists\, s (\overleftarrow{Tr} = s \wedge \lfloor (Tr - s) \nmid A = \langle\rangle \rfloor^*))[Ref - \{c_i?, c_i! \mid c_i \in A\}/Ref]$.

Theorem 3.20 (communication, concurrent composition and hiding)

(1) **let** $(x, t_1) = c?$ **in** P **end** $\|$ **let** $t_2 = c!r$ **in** Q **end** $\simeq$

let $(x, t_1) = c?$ **in** P $\|$ **let** $t_2 = c!r$ **in** $Q[t_2 + t_1/t_2]$ **end end**

$\square$**let** $t_2 = c!r$ **in** Q$\|$ **let** $(x, t_1) = c?$ **in** $P[t_1 + t_2/t_1]$ **end end**

$\square$**wait** 0; **let** x = r in $P[0/t_1] \| Q[0/t_2]$ **end**

(2) (**let** $(x, t_1) = c?$ **in** P **end** $\|$ **let** $t_2 = c!r$ **in** Q **end**)$\backslash\{c\} \simeq$

let x = r in $(P\lfloor 0/t_1 \rfloor \| Q\lfloor 0/t_2 \rfloor)\backslash\{c\}$ **end**

Interlocked composition. The interlocked composition P$\natural$Q is similar to concurrent composition, however, during the concurrent execution, any external communication is prevented. The semantics of interlocked composition is therefore defined similarly to that of concurrent composition, except using the redefined $[\![P]\!]^w_{ter}[i]$ and $[\![P]\!]^t_{ter}[i]$, using *Lock* instead of *Conc*, $LIdle^w$ and $LIdle^t$ instead of $CIdle^w$ and $CIdle^t$:

$$[\![P]\!]^w_{ter}[i] =_{df} ([\![P]\!]_{ter}[i] \wedge Lock) \circ LIdle^w(i)$$

$$[\![P]\!]^t_{ter}[i] =_{df} ([\![P]\!]_{ter}[i] \wedge Lock) \circ LIdle^t(i)$$

$$Lock =_{df} Maxp \wedge \exists\, s \cdot (\lfloor Tr = s \wedge Tr_0 = Tr_1 \rfloor^*)$$

$$LIdle^w(i) =_{df} \exists\, s_1, s_2, s_3 (\overleftarrow{Tr} = s \wedge \overrightarrow{Tr}_0 = s_0 \wedge \overrightarrow{Tr}_1 = s_1 \wedge \lfloor Tr - s = Tr_{i-1} - s_1 \rfloor \wedge Tr_i = s_i \wedge Ref = Ref_{i-1}\rfloor^*)$$

$$LIdle^t(i) =_{df} LIdle^w(i) \wedge \overleftarrow{V}_{a_j} = V^*_{a_j}$$

Note that we assume the concurrent or interlocked composition of P and Q can not cause infinite communications occurring in a finite interval, which is divergence. Therefore, for example, if P is an infinite loop of a single input, Q can not be an infinite loop of a single output on the same channel.

Theorem 3.21 (communication and interlocked composition)

let $(x, t_1) = c?$ **in** P **end** $\natural$ **let** $t_2 = c!r$ **in** Q **end** $\simeq$

let x = r in $P[0/t_1] \natural Q\lfloor 0/t_2 \rfloor$ **end**

4 Discussion

In this paper, we have given a denotational semantics to a subset of Timed RSL expressions using EDC with advanced features. Some algebraic laws of Timed RSL which can be proved from the denotational semantics are presented. To effectively reason about Timed RSL programs, we need a set of high-level syntax driven proof

rules. More case studies will be helpful.

Our work is a part of the Timed RAISE Project. RAISE is not only a specification language, it also includes the RAISE development method[20] and RAISE tools. The method of developing Timed RSL specifications and verification tools are also important research work for extending RAISE to Timed RAISE.

There has been some related work where semantics of several languages have been formalised using appropriate extensions of Duration Calculus. A semantics for an OCCAM-like language was defined in Ref. [17] using DC with super-dense chop, where the semantics was defined by two DC formulae, describing terminating and non-terminating behaviours respectively. Pandya, Wang and Xu[15] defined a compositional semantics of Sequential Hybrid Programs using Duration Calculus with super-dense time, fixed point operators and infinite intervals, where the semantics was defined by a single DC formula. They used a mixture of weakest and strongest fixed points to define the semantics of iteration.

In this paper, we define the semantics by a single DC formula, which is composed by three formulae describing divergence, waiting, and termination respectively. The semantics of iteration is defined by a single weakest fixed point. The advantage of our approach is that we can establish a link between the untimed refinement calculus and our timed one, which preserves the laws of untimed programming. In this paper, the super-dense chop is a derived modality, so that we can use the properties of the original chop to get a set of theorems for it.

References:

[1] Hale, R. W. S., Cardell-Oliver, R., Herbert, J. M. J. An embedding of timed transition system in HOL. Formal Methods in System Design, 1993,3:151~174.

[2] Maler, O., Manna, Z., Pnueli, A. From timed to hybrid systems. LNCS 600, 1992.

[3] Kohn, W. A declarative theory for rational controllers. In: Proceedings of the 27th CDC. 1988. 130~136.

[4] Nicolin, X., *et al*. An approach to the description and analysis of hybrid systems. LNCS 736, 1992,121~149.

[5] He, Ji-feng. From CSP to Hybrid Systems. In: Roscoe, A. W., ed. A Classical Mind. 1993. 171~190.

[6] Halpern, J., Manna, Z., Moszkowski, B. A hardware semantics based on temporal intervals. In: Proceedings of the 10th International Colloquium on Automata, Languages and Programming. 1983, 278~291.

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

[8] Hale, R. W. S. Programming in temporal logic. Technical Report 12 of Computing Laboratory, University of Cambridge, 1983.

[9] Moszkowski, B. C. Executing Temporal Logic Programs. Cambridge University Press, 1986.

[10] The RAISE Language Group: The RAISE Specification Language. Prentice Hall International (UK) Ltd., 1992.

[11] Zhou, Chao-chen, Ravn, A. P., Hansen, M. R. An extended duration calculus for hybrid systems. In: Hybrid Systems Grossman, R. L., Nerode, A., Ravn, A. P., *et al*., eds. Volume 736 of LNCS, Springer-Verlag, 1993. 36~59.

[12] Hansen, M., Pandya, P., Zhou, Chao-chen. Finite divergence. Theoretical Computer Science, 1995,138:113~139.

[13] Pandya, P., Ramakrishna, Y. A recursive duration calculus. Technical Report, CS-95/3, Computer Science Group, TIFR, Bombay, 1995.

[14] Pandya, P., Dang, V. H. Duration calculus with weakly monotonic time. Research Report 122, UNU/IIST, Macau, July 1996.

[15] Pandya, P., Wang, H. P., Xu, Q. W. Towards a theory of sequential hybrid programs. In: Gries, D., Willem-Paul de-Roever, eds. Programming Concepts and Methods (Proconet'98). Chapman & Hall, 1998, 366~384.

[16] Zhou, Chao-chen, Hoare, C. A. R., Ravn, A. P. A calculus of durations. Information Processing Letters, 1991,40(5): 269~276.

[17] Zhou, Chao-chen, Hansen, M. R. Chopping a point. In: He, J-F., Cooke, J., Willis, P., eds. Proceedings of the BCS FACS 7th Refinement Workshop, Electronic Workshops in Computing. Springer-Verlag, 1996.

[18] Chris, George, Xia, Yong. An operational semantics for timed RAISE. Research Report 149, UNU/IIST, Macau, November 1998.

[19] Tarski, A. A lattice theoretical fixpoint theorem and its applications. Pacific Journal of Mathematics, 1955,5:285~309.

[20] The RAISE Method Group. The RAISE Development Method. Prentice Hall International (UK) Ltd., 1995.

使用延时演算的时间化RSL的指称语义

李 黎[2], 何积丰[1,3]

[1](澳门联合国大学 国际软件技术研究所,澳门);

[2](中国科学技术大学,安徽 合肥 230026);

[3](华东师范学院,上海 200030)

摘要:使用扩展的持续时间演算(EDC)模型,给出了时间化的RAISE描述语言(RSL)的一个子集的指称语义.在扩展的持续时间演算模型中加入了一些新的特征,并探究了它们的代数定律.这些定律在形式化实时程序和验证实时性质中起着重要作用.最后还给出了时间化RSL的一些代数定律.这些定律可以从其指称语义证明,并用于程序的转化和优化.

关键词:延时演算;RAISE描述语言;指称语义;实时系统

中图法分类号:TP301 **文献标识码**:A