A Hybrid Relational Modelling Language

He Jifeng and Li Qin(✉)

Shanghai Key Laboratory of Trustworthy Computing,
International Research Center of Trustworthy Software,
East China Normal University, Shanghai, China
qli@sei.ecnu.edu.cn

Abstract. Hybrid systems are usually composed by physical components with continuous variables and discrete control components where the system state evolves over time according to interacting laws of discrete and continuous dynamics. Combinations of computation and control can lead to very complicated system designs. We treat more explicit hybrid models by proposing a hybrid relational calculus, where both clock and signal are present to coordinate activities of parallel components of hybrid systems. This paper proposes a hybrid relational modelling language with a set of novel combinators which support complex combinations of both testing and signal reaction behaviours to model the physical world and its interaction with the control program. We provide a denotational semantics (based on the hybrid relational calculus) to the language, and explore healthiness conditions that deal with time and signal as well as the status of the program. A number of small examples are given throughout the paper to demonstrate the usage of the language and its semantics.

Keywords: Formal language and semantics · Unifying theories of programming · Relation calculus · Hybrid systems

1 Introduction

Hybrid system is an emergent area of growing importance, emphasising a systematic understanding of dynamic systems that combine digital and physical effects. Combinations of computation and control can lead to very complicated system designs. They occur frequently in automotive industries, aviation, factory automation and mixed analog-digital chip design.

The basic conceptional definition of a hybrid system includes a direct specification of its behaviours associated with both continuous and discrete dynamics and their non-trivial interactions [dSS00, Bra95]. The states of hybrid systems evolve over time according to interacting laws of discrete and continuous dynamics. For

This work was supported by Shanghai Knowledge Service Platform Project (No. ZF1213), Doctoral Fund of Ministry of Education of China (No. 20120076130003) and the NSFC-Zhejiang Joint Fund for the Integration of Industrialization and Informatization (No. U1509219).

T. Gibson-Robinson et al. (Eds.): Roscoe Festschrift, LNCS 10160, pp. 124–143, 2017.
DOI: 10.1007/978-3-319-51046-0_7

discrete dynamics, the hybrid system changes state instantaneously and discontinuously; while during continuous transitions, the system state is a continuous function of continuous time and varies according to a differential equation.

Hybrid system modelers mix discrete time reactive systems with continuous time ones. Systems like Simulink treat explicit models made of *Ordinary Differential Equations*, while others like Modelica provide more general implicit models defined by *Differential Algebraic Equations*. A variety of models for hybrid systems have been developed, such as hybrid automata [ACH93, Hen96, Tav87], phase transition system [MMP91], declarative control [Koh88], extended state transition system [ZH04], and hybrid action systems [RRS03, Ben98]. Platzer proposed a logic called Differential Dynamic Logic for specifying properties of hybrid systems [Pla08, Pla10]. His hybrid systems analysis approach has also been implemented in the verification tool KeYmaera for hybrid systems. We refer the readers to [CPP06] for an overview of languages and tools related to hybrid systems modeling and analysis.

There are a number of specification languages developed for hybrid systems. Inspired by the work in [He94], Zhou et al. [ZWR96] presented a hybrid variant of *Communicating Sequential Processes* (HCSP) [Hoa85] as a language for describing hybrid systems. They gave a semantics in *the extended duration calculus* [ZH04]. Rönkkö et al. [RRS03] extended *the guarded command language* [Dij76] with differential relations and gave a weakest-precondition semantics in higher-order logic with built in derivatives. Rounds and Song [RS03] developed a hybrid version of the π-*calculus* [Mil99] as a modelling language for embedded systems. Modelling languages for hybrid systems further include *SHIFT* [Des96] for networks of hybrid automata, and *R-Charon* for reconfigurable systems [Kra06].

Rather than addressing the formal verification of hybrid systems using simulation based approaches or model checking, this paper focuses on a general framework. It uses a simple hybrid modelling language to model non-trivial interactions between hybrid dynamics. This language captures the defining features of the hybrid systems such as monitoring physical variables over continuous time, asynchronous reacting to control signals, etc. Following the *UTP* approach advocated in [HH98], we build a mathematical theory of the *hybrid relations* as the foundation of the hybrid modelling languages. This is a presentation within predicate calculus of Tarski's theory of relations [Tar41], enriched with his fixed point theory [Tar55]. We show that the hybrid relational calculus is a conservative extension of the classical relational calculus, *i.e.*, all the algebraic laws of the operators remain valid in the new calculus.

The rest of the paper is organised as follows.

The hybrid relational modelling language is proposed in Sect. 2. Its semantical model is provided in Sect. 3 with UTP approaches. Section 3.1 adds continuous variables into the alphabet of relations to record the continuous dynamic behaviors of the hybrid system.

In Sect. 3.2, healthiness conditions placed on hybrid relations are proposed to ensure that the hybrid relations satisfy additional desirable properties related to clocks, signals and intermediate observations between initiation and termination. Sections 3.3 to 3.5 give a denotational semantics to every primitive command and combinator in the hybrid modelling language including the concurrent composition and the novel synchronous constructs **until** and **when** proposed to specify the interactions between components.
The paper ends with Sect. 4 for conclusion and future works.

2 A Hybrid Modelling Language

This section presents a hybrid modelling language, which extends the guarded command language [Dij76] by adding output command, synchronisation constructs and parallel operator. The syntax of the hybrid modelling language is as follows where x is a discrete variable, v is a continuous variable and s is a signal.

$$
\begin{aligned}
AP &::= \mathbf{skip} \mid \mathbf{chaos} \mid \mathbf{idle} \mid x := e \mid x \leftarrow v \mid !s \mid \mathbf{delay} \mid \mathbf{delay}(\delta) \\
EQ &::= R(v, \dot{v}) \mid EQ \ \mathbf{init}\ (v = e) \mid EQ|EQ \\
P &::= AP \mid P \sqcap P \mid P; P \mid P \lhd b(x) \rhd P \mid P \| P \mid \mu X \bullet P(X) \mid \\
&\quad EQ \ \mathbf{until}\ g \mid \mathbf{when}(G) \mid \mathbf{timer}\ c \ \bullet P \mid \mathbf{signal}\ s \ \bullet \ P \\
g &::= \mathbf{I} \mid signal \mid test \mid g \cdot g \mid g + g \\
test &::= \mathbf{true} \mid v \geq e \mid v \leq e \mid \neg test \mid test \wedge test \mid test \vee test \\
G &::= g \& \ P \mid G [\!] G
\end{aligned}
$$

AP is a collection of atomic commands. **skip** is an atomic program that terminates immediately without changing any state value. **chaos** is an atomic program that diverges immediately. **idle** is an atomic program that never terminates and does not send out signals. $x := e$ is the conventional assignment which assigns the value of a discrete expression e to a discrete variable x. $x \leftarrow v$ samples the current value of a continuous variable v and assigns it to a discrete variable x. $!s$ emits a signal s. **delay** acts like **skip** but its terminating time is unknown in advance. $\mathbf{delay}(\delta)$ keeps idle and terminates after δ time units.
EQ contains statements for continuous dynamics. $R(v, \dot{v})$ is a differential relation specifying the dynamics of the continuous variable v. EQ **init** v_0 assigns the initial value v_0 to the continuous variable v governed by EQ. $EQ|EQ$ is a conjunction of two dynamics.
P lists all combinators in the hybrid language. The first line includes classic sequential composition operators, parallel composition operators and recursion operator. The first two structures in the second are new hybrid structures specifying the interactions between the continuous and discrete components of the hybrid system. They will be introduced in detail in Sect. 3.5. The last two operations of P are hiding operators for timers and signals.

The last three lines of the syntax comprise the structure of a guard command language for G which is a core element of the new hybrid structures. The guard condition g can be a signal, a value test and their combination. The notation $\mathbf{I}$ stands for a guard condition which will always be triggered immediately. $g\&P$ is a reactive structure that will execute P when g is triggered. $G[\!]G$ stands for a guarded choice operator.

Example 1 (Temperature control system). Consider a simple hybrid system controlling the temperature of a room. We use a continuous variable θ to record the room temperature and a discrete variable $H : \{on, off\}$ to denote the status of the heater. When the room temperature is below 19 degrees, the heater will be turned on and when the temperature exceeds 20, the heater will be turned off. Let Δ be the changing rate of the temperature, the specification of such system in our modelling language is as follows.

$$H := off; (\dot{\theta} = -\Delta \textbf{ init } \theta = 25) \textbf{ until } (\theta \leq 19);$$

$$\textbf{when } \begin{pmatrix} \theta \leq 19 \ \& \ (H := on; (\dot{\theta} = \Delta) \textbf{ until } \theta \geq 20) \ [\!] \\ \theta \geq 20 \ \& \ (H := off; (\dot{\theta} = -\Delta) \textbf{ until } \theta \leq 19) \end{pmatrix}^*$$

where P^* stands for the recursive program $\mu X \bullet (P; X)$.

3 Semantical Model

The semantics of the hybrid language is defined based on the UTP theory. We will first choose the alphabet and healthiness conditions for the hybrid programs and provide the denotational semantics for every command and combinator. We refer the readers to [HH98] for the basic notations of UTP theory. And for the lack of space, we omit all proofs of the theorems in this section.

3.1 Alphabet

The hybrid programs studied in this paper are formalised with hybrid relations with an enlarged alphabet including continuous variables.

Definition 1 (Hybrid Relation).
A hybrid relation *is a pair* $(\alpha P, P)$*, where* P *is a predicate containing no free variables other than in* αP*. Its alphabet* αP *contains sets of input and output discrete variables and a set* $\mathbf{con}\alpha P$ *of* continuous variables.

$$\alpha P \ = \ \mathbf{in}\alpha P \ \cup \ \mathbf{out}\alpha P \ \cup \ \mathbf{con}\alpha P$$

The input variable set $\mathbf{in}\alpha P \ =_{df} \ \{st, t, pos\} \ \cup \ PVar \ \cup \ ClockVar$

where
$st, st' \in \{term, stable, div\}$ represent the program status at its start and finish time respectively. The meanings of program status $term, stable, div$ are introduced in Sect. 3.2.

t, $t' \in Time$ (of the type non-negative real numbers) are discrete variables denoting the start and end time instances of the observation one makes during the execution of the program.

$pos : \mathbb{N}$ (of the type nature numbers) is a variable introduced to facilitate the mechanism for describing the dependency of the signals. Its value will be recorded in the clock of each signal when it is emitted. The detailed usage will be demonstrated later associated with the clock variables.

$PVar$ denotes the set of discrete program variables.

$ClockVar$ denotes the set of clock variables

$$ClockVar =_{df} \{s.\mathbf{clock} \mid s \in \mathbf{InSignal} \cup \mathbf{OutSignal}\}$$

where **InSignal** and **OutSignal** stand for the sets of input signals and output signals respectively with the constraint $\mathbf{InSignal} \cap \mathbf{OutSignal} = \emptyset$.
A clock variable $s.\mathbf{clock}$ is a sequence of pairs with the type $Time \times \mathbb{N}$. For a pair (τ, p) as an entry of the sequence, τ stands for the time instant at which s occurs, while p denotes its emitting position in the queue of the dependent signals that are observed at the same instant. For example, if the emission of s_2 depends on the emission of s_1 at time τ, then we have $(\tau, m) \in s_1.\mathbf{clock}$ and $(\tau, n) \in s_2.\mathbf{clock}$ with $m < n$.

The output alphabet contains the dashed variables from input alphabet.

$$\mathbf{out}\alpha P = \{x' \mid x \in \mathbf{in}\alpha P\}$$

The continuous variables in $\mathbf{con}\alpha P$ are mappings from time to corresponding physical status of the physical components, *i.e.*, of the type $Time \to Real$. The set $\mathbf{con}\alpha P = \mathbf{own}\alpha P \cup \mathbf{env}\alpha P$ is divided into two sets: $\mathbf{own}\alpha P$ and $\mathbf{env}\alpha P$ where the former comprises those continuous variables owned by the relation, and the latter denotes the set of variables that are accessible by P but managed by the environment. The set $\mathbf{own}\alpha P = \mathbf{phy}\alpha P \cup \mathbf{timer}\alpha P$ includes a special subset $\mathbf{timer}\alpha P$ to specify the timers owned by P.
A refinement order can be defined over hybrid relations as follows.

Definition 2 (Refinement).
Let P and Q be hybrid relations with the same alphabet A. We will use the notation $P \sqsupseteq Q$ to abbreviate the formula $\forall x, y, ..., u, v \bullet (P \Rightarrow Q)$ where $x, y, ..., u, v$ are all the variables of the alphabet A.

3.2 Healthiness Conditions

In this section, we will introduce healthiness conditions one by one and show that every healthiness condition obtains a subset of the previous domain and the healthy programs form a complete lattice w.r.t. the refinement order.

Time.
To describe the dynamical behaviour of physical components we will focus on

those hybrid relations in which the termination time is not before its initial time. As a result, we require a hybrid relation P to meet the following healthiness condition:

$(\mathbf{HC1})P \;=\; P \wedge (t \leq t')$

A hybrid relation is called **HC1**-healthy if it satisfies the condition **HC1**.
We introduce a function **H1** to turn a hybrid relation into a **HC1**-healthy hybrid relation:

$\mathbf{H1}(P) \;=_{df}\; P \wedge (t \leq t')$

It is trivial to show that **H1** is monotonic and idempotent.

Signals.
Signals are means of communications and synchronisations between different components and between a program with its environment. In general, a signal, denoted by its name, has two types of status, *i.e.*, either presence or absence. A signal is present if it is received by a program from its environment, or it is emitted as the result of an output command.
For any signal s, we use a clock variable $s.\mathbf{clock}$ to record the time instants at which s has been present. As usual, we adopt $s.\mathbf{clock}$ and $s.\mathbf{clock}'$ to represent the values at the start time t and the finish time t' correspondingly. $s.\mathbf{clock}$ has to be a subset of $s.\mathbf{clock}'$ since the latter may be added some time instants of $[t, t']$ at which the signal s is present. Consequently, we require a hybrid relation P to meet the following healthiness condition:

$(\mathbf{HC2})P \;=\; P \;\wedge\; \mathbf{inv}(s)$

where $\mathbf{inv}(s) \;=_{df}\; (s.\mathbf{clock} \subseteq s.\mathbf{clock}') \wedge (s.\mathbf{clock}' \subseteq (s.\mathbf{clock} \cup [t, t'] \times \mathbb{N}))$

$\mathbf{H2}(P) \;=_{df}\; P \wedge \mathbf{inv}(s)$

It is trivial to prove that the order in which **H1** and **H2** are composed is irrelevant, *i.e.*, $\mathbf{H1} \circ \mathbf{H2} \;=\; \mathbf{H2} \circ \mathbf{H1}$. With this fact, we can define a composite mapping $\mathbf{H12} \;=_{df}\; \mathbf{H1} \circ \mathbf{H2}$. And it can be proved that **HC1** and **HC2**-healthy hybrid relations are closed under choice, conditional and sequential composition.

Theorem 1.

(1) $\mathbf{H12}(P) \sqcap \mathbf{H12}(Q) = \mathbf{H12}(P \sqcap Q)$
(2) $\mathbf{H12}(P) \lhd b \rhd \mathbf{H12}(Q) \;=\; \mathbf{H12}(P \lhd b \rhd Q)$
(3) $\mathbf{H12}(P); \mathbf{H12}(Q) \;=\; \mathbf{H12}(\mathbf{H12}(P); \mathbf{H12}(Q))$

For simplicity, we will confine ourselves to **HC1** *and* **HC2***-healthy hybrid relations in the next section.*

Intermediate Observation and Divergence.
In this section, we add logical variables st and st' to the input alphabet and the output alphabet of a hybrid relation to describe the program status before it

starts, and the status it completes respectively. These variables range over the set $\{term, stable, div\}$, where
$st = term$ indicates the predecessor of the hybrid program terminates successfully. As a result, the control passes to the current hybrid program.
$st = stable$ indicates its predecessor is waiting for ignition. Therefore, the hybrid program can not start its execution because its predecessor has not finished yet.
$st = div$ indicates the behaviour of the predecessor becomes chaotic, and it can not be rescued by the execution of the current hybrid program.
Here we propose an order $<$ over the set of program status:

$$div \ < \ stable \ < \ term$$

This order can be adopted to define the merge mechanism for the parallel composition operator in Sect. 3.5.

Example 2 (Atomic Hybrid Relations).

Let $PVar$ be a set of discrete data variables, and

$$A \ =_{df} \ \{st, t, pos\} \ \cup \ PVar \ \cup \ \{s.\mathbf{clock} \mid s \in \mathbf{OutSignal}\}$$

(1) The hybrid relation **skip** does nothing, and terminates immediately.
$$\mathbf{skip} \ =_{df} \ I\!I_A \lhd (st \neq div) \rhd \mathbf{H12}(\bot_A)$$
where $I\!I_A$ is the identity relation over set A and $\bot_A =_{df} true$.
(2) **chaos** represents the worst hybrid program, and its behaviour is totally unpredictable.
$$\mathbf{chaos} \ =_{df} \ \mathbf{H12}(\bot_A) \lhd st = term \rhd \mathbf{skip}$$
(3) **delay** behaves like hybrid program **skip** except its termination time is unknown in advance.
$$\mathbf{delay} \ =_{df} \ \mathbf{H12}(I\!I_{A\setminus\{t\}}) \lhd st = term \rhd \mathbf{skip}$$

From Theorem 1 it follows that these atomic hybrid programs are **HC1** and **HC2** healthy. Note that the above atomic hybrid relations have no constraints to the continuous variables. □

The healthiness conditions relevant to st are proposed to capture the intermediate observation (**HC3**) and divergence (**HC4**) features of hybrid programs.
A hybrid program P remains idle until its sequential predecessor terminates successfully. This constraint requires P to satisfy the following healthiness condition:

$$(\mathbf{HC3}) \ P \ = \ P \lhd st = term \rhd \mathbf{skip}$$

We can prove that all atomic hybrid programs of Example 2 are **HC3**-healthy.

$$\mathbf{H3}(P) \ =_{df} \ P \lhd st = term \rhd \mathbf{skip}$$

A **HC3**-healthy program has **skip** as its left unit and **chaos** as its left zero.

Theorem 2 (Left unit and left zero).

(1) $\mathbf{skip}; \mathbf{H3}(P) \ = \ \mathbf{H3}(P)$
(2) $\mathbf{chaos}; \mathbf{H3}(P) \ = \ \mathbf{chaos}$

Once a hybrid program enters a divergent state, its future behaviour becomes uncontrollable. This requires it to meet the following healthiness condition:

(**HC4**) $P = P; \mathbf{skip}$

$\mathbf{H4}(P) =_{df} P; \mathbf{skip}$

HC4-healthy programs are closed under choices, conditional and sequential composition.

Theorem 3.

(1) $\mathbf{H4}(P) \sqcap \mathbf{H4}(Q) = \mathbf{H4}(P \sqcap Q)$
(2) $\mathbf{H4}(P) \lhd b \rhd \mathbf{H4}(Q) = \mathbf{H4}(P \lhd b \rhd Q)$
(3) $\mathbf{H4}(P); \mathbf{H4}(Q) = \mathbf{H4}(P; Q)$ *provided that* Q *is* **HC3***-healthy.*

The composition order of **H3** and **H4** is irrelevant, *i.e.*, $\mathbf{H4} \circ \mathbf{H3} = \mathbf{H3} \circ \mathbf{H4}$. Define $\mathbf{H} =_{df} (\mathbf{H1} \circ \mathbf{H2} \circ \mathbf{H3} \circ \mathbf{H4})$. We can prove that **H** is monotonic and idempotent and $\mathbf{H} = \mathbf{H3} \circ \mathbf{H4}$.
The mapping **H** distributes over non-deterministic choice, conditional and sequential composition.

Theorem 4.

(1) $\mathbf{H}(P) \sqcap \mathbf{H}(Q) = \mathbf{H}(P \sqcap Q)$
(2) $\mathbf{H}(P) \lhd b \rhd \mathbf{H}(Q) = \mathbf{H}(P \lhd b \rhd Q)$
(3) $\mathbf{H}(P); \mathbf{H}(Q) = \mathbf{H}(\mathbf{H}(P); \mathbf{H}(Q))$

The distributivity of **H** over parallel operators will be shown in Sect. 3.5. To summarize, the healthy hybrid program domain is closed under these composition operators.

Theorem 5.
The domain of healthy hybrid programs $\mathbb{P} =_{df} \{P \mid P = \mathbf{H}(P)\}$ *and the refinement order* $\sqsupseteq$ *forms a complete lattice* $L =_{df} (\mathbb{P}, \sqsupseteq)$.

3.3 Atomic Commands

The definitions of atomic commands **skip**, **chaos** and **delay** are already given in *Example 2*. One can verify that they are all healthy w.r.t. the mapping **H**.
Let e be an expression with only discrete variables. Assignment $x := e$ assigns the value of e to the discrete variable x instantaneously. It supports the discrete state change of the hybrid programs.

$(x := e) =_{df} \mathbf{H}(\Pi_{\mathbf{in}\alpha}[e/x])$

Let v be a continuous variable in $\mathbf{own}\alpha$. Assignment $x \leftarrow v$ assigns the current value of v to the discrete variable x instantaneously. It provides a direct way in the language for sampling the value of a continuous variable to a discrete program variable.

$(x \leftarrow v) =_{df} \mathbf{H}(\Pi_{\mathbf{in}\alpha \setminus \{x\}} \wedge x' = v(t'))$

The output command $!s$ emits signal s, and then terminates immediately. Its

execution does not take time.

$$!s \ =_{df} \ \mathbf{H}(\Pi_{\mathbf{in}\alpha}[(s.\mathbf{clock} \cup \{(t,\, pos)\})/s.\mathbf{clock}])$$

The program **idle** never terminates, and keeps stable status forever.

$$\mathbf{idle} \ =_{df} \ \mathbf{H}(\Pi_B \ \wedge \ \mathbf{time{-}passing} \ \wedge \ st' = stable)$$

where

$$B =_{df} \{s.\mathbf{clock} \mid s \in \mathbf{OutSignal}\}$$

$$\mathbf{time{-}passing} =_{df} \bigwedge_{c \in \mathbf{timer}\alpha} \forall \tau \in [t,\, t') \bullet (\dot{c}(\tau) = 1)$$

Let $\delta \ \geq \ 0$. The delay command $\mathbf{delay}(\delta)$ suspends the execution δ time units.

$$\mathbf{delay}(\delta) \ =_{df} \ \mathbf{H}\left(\begin{array}{l} \Pi_B \ \wedge \ \mathbf{time{-}passing} \wedge \\ \left(\begin{array}{l} (t'-t) < \delta \ \wedge \ st' = stable \ \vee \\ (t'-t) = \delta \ \wedge \ \Pi_{\{pos\} \cup PVar} \ \wedge \ st' = term \end{array}\right) \end{array}\right)$$

Notice that the difference between **delay** and $\mathbf{delay}(\delta)$ is that the end time t' of **delay** is unspecified (arbitrarily after its start time).

3.4 Dynamics of Continuous Variables

Let v be a continuous variable used to model the status of a physical device. The continuous transitions of v governed by the physical laws can usually be specified by a hybrid relation $R(v,\, \dot{v})$, whose dynamic behaviour over an interval $[t,\, t']$ is described by

$$R \ =_{df} \ \forall \tau \in [t,\, t') \bullet \ R(v,\, \dot{v})(\tau)$$

Let e be an expression with only discrete variables. The hybrid relation R **init** $(v = e)$ sets the value of e as the initial value of continuous variable v.

$$R \ \mathbf{init} \ (v = e) \ =_{df} \ R \ \wedge \ (v(t) \ = \ e)$$

Let R_1 and R_2 be hybrid relations of distinct variables v and w. Their composition $R_1 \mid R_2$ is simply defined as the conjunction of R_1 and R_2:

$$R_1 \mid R_2 \ =_{df} \ R_1 \ \wedge \ R_2$$

Differential equation $\dot{v} = f(v)$ and differential-algebraic equation $(F(v,\, \dot{v}, t) = 0)$ are both seen as a special kind of hybrid relations over continuous variable v.

Example 3. Let v be a continuous variables over continuous time c. A differential-algebraic equation $F(v,\, \dot{v},\, c) = 0$ can be treated as a hybrid relation where

$$DF =_{df} \ (t \leq t') \ \wedge \ \forall \tau : [t,\, t') \bullet \ (F(v(\tau),\, \dot{v}(\tau),\, \tau) \ = \ 0) \qquad \square$$

The refinement order defined for hybrid relations in Sect. 3.1 can be applied to the relation R.

Definition 3.
Assume that $R_1(v,\, \dot{v})$ and $R_2(v,\, \dot{v})$ are equipped with the same alphabet (say $\{v\}$), we define

$$R_1 \sqsupseteq R_2 \ =_{df} \ \forall t, t', \forall v \bullet (R_1 \Rightarrow R_2)$$

It means that if a continuous variable v is a solution of R_1, then it is also a solution of R_2. In other terms, R_1 can be considered as a refinement of R_2 since any continuous evolvement it allows for the continuous variable v is also allowed by R_2.

3.5 Combinators

Let P and Q be hybrid programs, the combinators of the hybrid language include the classic sequential operators, parallel operators and recursion operators. Besides, it has two hybrid reactive structures specifying the interactions between the continuous and discrete components of the system. In this section, we will give the definitions of the combinators.

Sequential Operators.
The sequential programming operators, including nondeterministic choice $P \sqcap Q$, conditional choice $P \lhd b \rhd Q$ and sequential composition $P; Q$ can be defined by the same predicates as in [HH98] but over the enriched alphabet for hybrid relations satisfying healthiness conditions. For lack of space, we only give the definition of $P; Q$ for example.

Definition 4 (Sequential Composition).
Let P and Q be hybrid relations with $\mathbf{out}\alpha P = \{x' \mid x \in \mathbf{in}\alpha Q\}$, $\mathbf{own}\alpha P = \mathbf{own}\alpha Q$ *and* $\mathbf{env}\alpha P = \mathbf{env}\alpha Q$. *The sequential composition $P; Q$ is defined by the following predicate:*

$$P; Q =_{df} \exists m \bullet P[m/x'] \wedge Q[m/x]$$

with $\alpha(P; Q) =_{df} \mathbf{in}\alpha P \cup \mathbf{out}\alpha Q \cup \mathbf{con}\alpha P$.

The sequential composition operator enjoys the same set of algebraic laws as its counterpart given in [HH98].

Parallel Operators.
Before we get to the definition of the parallel composition of hybrid programs, we first revisit two notions of parallel composition operators that will be employed in the definition.

Definition 5 (Disjoint Parallel Operator).
Let P and Q be hybrid relations with disjoint $\mathbf{out}\alpha$ *and* $\mathbf{own}\alpha$. *The notation $P \mid Q$ represents the following hybrid relation*

$$P \mid Q =_{df} P \wedge Q$$

with $\mathbf{in}\alpha(P \mid Q) =_{df} \mathbf{in}\alpha P \cup \mathbf{in}\alpha Q$, $\mathbf{own}\alpha =_{df} \mathbf{own}\alpha P \cup \mathbf{own}\alpha Q$ *and* $\mathbf{env}\alpha =_{df} (\mathbf{env}\alpha P \setminus \mathbf{own}\alpha Q) \cup (\mathbf{env}\alpha Q \setminus \mathbf{own}\alpha P)$.

The operator $\mid$ is symmetric and associative. It distributes over conditional, and has $\Pi_\emptyset$ as its unit. Moreover, $\mid$ and ; satisfy the mutual distribution law.
For programs whose $\mathbf{out}\alpha$ and $\mathbf{own}\alpha$ are overlapped, we employ a parallel by merge operator to merge the results of the parallel components.

Definition 6 (Merge Mechanism).
A merge mechanism M is a pair $(x : \mathbf{Val}, op)$, where x is a variable of type **Val**, *and op is a binary operator over* **Val**.

Definition 7 (Parallel by Merge).
Let P and Q be hybrid relations with the shared output x' and its merge mechanism $M = (x : \mathbf{Val}, op)$. We define their parallel composition by merge, denoted $P \parallel_M Q$, as follows:

$$P \parallel_M Q =_{df} \exists m, n : \mathbf{Val} \bullet \begin{pmatrix} P[m/x'] \ \wedge \ Q[n/x'] \ \wedge \\ x' \ = \ op(m, n) \end{pmatrix}$$

with $\mathbf{in}\alpha(P \parallel_M Q) =_{df} \mathbf{in}\alpha P \cup \mathbf{in}\alpha Q$, $\mathbf{own}\alpha(P \parallel_M Q) =_{df} \mathbf{own}\alpha P \cup \mathbf{own}\alpha Q$ *and* $\mathbf{env}\alpha(P \parallel_M Q) =_{df} (\mathbf{env}\alpha P \setminus \mathbf{own}\alpha Q) \cup (\mathbf{env}\alpha Q \setminus \mathbf{own}\alpha P)$.

With the above notions of parallel operator, we can define the semantics of general parallel composition $P \parallel Q$. We need to merge the program status *st* and the *pos* variables from both components.

For *st*, we select the merge operator for the program status as the greatest lower bound, *i.e.*, **glb** (remember that we have the order $div < stable < term$).

For *pos*, we select the merge operator as **max** which selects the greater value.

Definition 8 (Parallel Operator for Hybrid Programs).

Let P and Q be hybrid programs satisfying the following conditions:

$PVar(P) \cap PVar(Q) = \emptyset$, $\mathbf{own}\alpha(P) \cap \mathbf{own}\alpha(Q) = \emptyset$,

$\mathbf{timer}\alpha(P) \cap \mathbf{env}\alpha(Q) = \emptyset$, $\mathbf{timer}\alpha(Q) \cap \mathbf{env}\alpha(P) = \emptyset$ *and*

$\mathbf{OutSignal}(P) \cap \mathbf{OutSignal}(Q) = \emptyset$

The parallel composition $P \parallel Q$ is equipped with the following alphabet:

$PVar =_{df} PVar(P) \ \cup \ PVar(Q)$, $\mathbf{phy}\alpha =_{df} \mathbf{phy}\alpha(P) \ \cup \ \mathbf{phy}\alpha(Q)$,

$\mathbf{timer}\alpha =_{df} \mathbf{timer}\alpha(P) \ \cup \ \mathbf{timer}\alpha(Q)$,

$\mathbf{env}\alpha =_{df} (\mathbf{env}\alpha(P) \setminus \mathbf{own}\alpha(Q)) \cup (\mathbf{env}\alpha(Q) \setminus \mathbf{own}\alpha(P))$,

$\mathbf{InSignal} =_{df} (\mathbf{InSignal}(P) \setminus \mathbf{OutSignal}(Q)) \cup (\mathbf{InSignal}(Q) \setminus \mathbf{OutSignal}(P))$,

$\mathbf{OutSignal} =_{df} \mathbf{OutSignal}(P) \ \cup \ \mathbf{OutSignal}(Q)$.

The dynamic behaviour of $P \parallel Q$ is described by

$$P \parallel Q \ =_{df} \ (((P; \mathbf{delay}) \parallel_M Q) \ \vee \ (P \parallel_M (Q; \mathbf{delay})))); \mathbf{skip}$$

where the merge mechanism M is defined by

$$M \ =_{df} \ ((st, pos) : \begin{pmatrix} (\{term, stable, div\}, \mathbb{N}), \\ (\mathbf{glb}, \mathbf{max}) \end{pmatrix}$$

The **delay** commands are used to synchronise the end time of the two components and the successive **skip** command makes the program satisfy **HC4**. The

merge mechanism M merges the status of the parallel components with a greatest lower bound operator. For example, if st' of P is *term* and st' of Q is *stable*, then the st' of $P \parallel Q$ is *stable*. It also merges the pos' for the output signals to be the greater one, *i.e.* if pos' of P is m and pos' of Q is n, then the pos' of $P||Q$ is $\mathbf{max}(m, n)$.

With the definition of the merge mechanism M, we can obtain that the domain of healthy hybrid relations is closed w.r.t. parallel composition.

Theorem 6.
If P and Q is healthy hybrid relations, i.e., $P = \mathbf{H}(P)$ *and* $Q = \mathbf{H}(Q)$, *then so does* $P\|_M Q$, i.e., $P\|_M Q = \mathbf{H}(P\|_M Q)$.

The parallel composition is symmetric and associative, and distributes over conditional and nondeterministic choices. Furthermore, it has **skip** and **chaos** as its unit and zero respectively. Moreover, the parallel composition has a true concurrent semantics: parallel components proceed independently and simultaneously.

Theorem 7.

(1) $(x := e; P) \parallel Q = (x := e); (P \parallel Q)$
(2) $(\mathbf{delay}(\delta); P) \parallel (\mathbf{delay}(\delta); Q) = \mathbf{delay}(\delta); (P \parallel Q)$
(3) $\mathbf{delay}(\epsilon) \parallel \mathbf{delay}(\delta) = \mathbf{delay}(\mathbf{max}(\epsilon, \delta))$

Guard Condition.
This subsection focusses on the guard conditions that will appear in the new hybrid structures **when** and **until** which will be defined in the next section.
In our hybrid language, the guard condition supports the following form: (1) value test: monitoring the value changing of a continuous variable. (2) signal: monitoring the emission of a signal. (3) their combinations via operator $\cdot$ and $+$. Like the hybrid automata, our language supports a transition when the value of a continuous variable exceeds a given bound. In addition, our language can support the reactions to receiving certain signals from the environment.

Example 4 (Gear shifting). Consider a car-driving control system. For a manual transmission car, its accelerating process can be divided into 4 shifting modes depending on the running gears. Assume that the proper speed interval for shifting from gear 1 to gear 2 is 20 kph to 30 kph. The car will change gear from 1 to 2 when (1) the current speed lies in the interval, **and** (2) the driver pushes the gear lever from 1 to 2. Let signal *gear_up* means the driver pushes the gear lever, the specification of the shifting can be written as follows.

$$\mathbf{when}\ ((20 < v \leq 30) \cdot gear\_up\ \&\ Gear_2)$$

where v is a continuous variable representing the speed of the car; $Gear_2$ represents the specification for the running mode of the car with gear 2. The guard condition involves both value testing and signal reception. □

To specify the reactive behaviours, we need to define the trigger condition of the guards. We introduce the following function **fired** to specify the status of a guard g over time interval:

$$g.\mathbf{fired} : Interval \to Time \to Bool$$

where for any $\tau \in [t, t']$, $g.\mathbf{fired}([t, t'])(\tau) = true$ indicates g is fired at the time instant τ. In other terms, given a time instant τ within the time interval $[t, t']$, the function tells us whether the guard g is fired at τ.
This function is defined by induction as follows:

1. **I** is ignited immediately after it starts its execution.
   $\mathbf{I}.\mathbf{fired}([t, t'])(\tau) =_{df} (\tau = t)$
2. s is fired whenever an input signal s is received.
   $s.\mathbf{fired}([t, t'])(\tau) =_{df} \exists n \in \mathbb{N} \bullet (\tau, n) \in s.\mathbf{clock}'$
3. *test* is fired whenever the value of expression *test* is true at that time instant.
   $test.\mathbf{fired}([t, t'])(\tau) =_{df} test(\tau)$
4. the composite guard $g_1 \cdot g_2$ is fired only when both g_1 and g_2 are fired simultaneously.
   $(g_1 \cdot g_2).\mathbf{fired} =_{df} g_1.\mathbf{fired} \wedge g_2.\mathbf{fired}$
5. the composite guard $g_1 + g_2$ is fired when either g_1 or g_2 is fired.
   $(g_1 + g_2).\mathbf{fired} =_{df} g_1.\mathbf{fired} \vee g_2.\mathbf{fired}$

Two guards are identical if they have the same firing function:
$(g = h) =_{df} (g.\mathbf{fired} = h.\mathbf{fired})$

From the above definitions and the properties of predicate combinators we conclude that both guard combinators $\cdot$ and $+$ are idempotent, symmetric and associative, and furthermore $\cdot$ distributes over $+$.

Theorem 8.

(1) $(Guard, +, \cdot, \mathbf{true}, \mathbf{false})$ *forms a Boolean algebra.*
(2) $g + \mathbf{true} = \mathbf{true}$.
(3) $g \cdot \mathbf{false} = \mathbf{false}$.

We say g is weaker than h (denoted by $g \leq h$), if the ignition of h can awake g immediately:
$g \leq h =_{df} h = (h \cdot g)$
From Theorem 8(1) it follows that $\leq$ is a partial order. Then we have
$g \leq h$ iff $g = (g + h)$.
In order to specify the blocking behaviour of the **when** construct, we need to define a trigger condition for the guard condition so that it is fired at the endpoint of a time interval and before that it remains unfired. To identify such cases we introduce the boolean function $g.\mathbf{triggered} : Interval \to Bool$.

$$g.\mathbf{triggered}([t, t']) =_{df} \begin{pmatrix} g.\mathbf{fired}([t, t'])(t') \wedge \\ \forall \tau \in [t, t') \bullet \neg g.\mathbf{fired}([t, t'])(\tau) \end{pmatrix}$$

To specify those cases when the guard g remains inactive we introduce the boolean function $g.\mathbf{inactive} : Interval \rightarrow Bool$.

$$g.\mathbf{inactive}([t, t']) \ =_{df} \ \forall \tau \in [t, t'] \bullet \neg g.\mathbf{fired}([t, t'])(\tau)$$

Note that $g.\mathbf{triggered} \neq \neg g.\mathbf{inactive}$. For example, let g be $(c = 3)$ where c is a timer. For the interval $[0, 4]$, we have both $g.\mathbf{triggered} = \mathbf{false}$ and $g.\mathbf{inactive} = \mathbf{false}$ since $g.\mathbf{fired}([0, 4])(3) = \mathbf{true}$.
The corresponding boolean functions for the composition of guards have the following properties.

Theorem 9.

$$\textit{(1)} \ (g_1 + g_2).\mathbf{triggered} \ = \ \begin{pmatrix} g_1.\mathbf{triggered} \wedge (g_2.\mathbf{triggered} \vee g_2.\mathbf{inactive}) \ \vee \\ g_2.\mathbf{triggered} \wedge (g_1.\mathbf{triggered} \vee g_1.\mathbf{inactive}) \end{pmatrix}$$

$$\textit{(2)} \ (g_1 + g_2).\mathbf{inactive} = \ g_1.\mathbf{inactive} \ \wedge \ g_2.\mathbf{inactive}$$

When Statement.
With the boolean functions **triggered** and **inactive** defined above, we can define the semantics of the **when** construct.
The program $\mathbf{when}(g_1 \& P_1 [\!] ....[\!] g_n \& P_n)$ waits for one of its guards to be fired, then selects a program P_i with the ignited guard to be executed. It is much like the conventional guarded choice construct except that its guards refer to continuous variables and signals whose status change through time.
In detail, the behaviour of $\mathbf{when}(g_1 \& P_1 [\!] ....[\!] g_n \& P_n)$ can be interpreted as follows.

(1) It will keep waiting $(st' = stable)$ when every guard is inactive in its execution interval $[t, t']$.
(2) It will execute P_i when g_i is triggered during its execution interval $[t, t']$. If more than one guard is triggered, the triggered branches are selected nondeterministically.

Definition 9.

$$\mathbf{when}(g_1 \& P_1 [\!] ....[\!] g_n \& P_n) =_{df}$$

$$\mathbf{H}(st' = stable \wedge II_B \wedge \mathbf{time-passing} \ \wedge \ (g_1 + ... + g_n).\mathbf{inactive})) \ \vee$$

$$\bigvee_{1 \leq i \leq n} \mathbf{H} \begin{pmatrix} st' = term \wedge II_{PVar \cup B} \wedge \ \mathbf{time-passing} \ \wedge \ \mathbf{update}(pos, g_i) \wedge \\ g_i.\mathbf{triggered} \ \wedge \ (g_1 + .. + g_n).\mathbf{triggered} \end{pmatrix} ; P_i$$

where

$$B =_{df} \{s.clock \mid s \in \mathbf{OutSignal}\}$$

$$\mathbf{update}(pos, g) =_{df} (pos' = pos) \lhd g \cap Signal \ = \ \emptyset \rhd (pos' > \mathbf{max}(pos, \mathbf{index}(g))$$

$$\mathbf{index}(g) =_{df} \ \mathbf{max}(\{0\} \ \cup \ \{\pi_2(\mathbf{last}(s.clock')) \mid s \in g\})$$

The $\mathbf{update}(pos, g)$ makes sure that the variable pos' records the maximum index of the signals emitted at the same time so far.
From Theorem 4 we conclude that $\mathbf{when}(g_1 \& P_1 [\!] ... [\!] g_n \& P_n)$ lies in the complete lattice L introduced in Theorem 5 whenever all guarded programs P_i are elements of L. In other words, the healthy hybrid relation domain is closed w.r.t. the **when** construct.

Some interesting algebraic laws of the **when** statement are listed below:

Theorem 10.

(1) The guards of the same guarded branch can be composed by + operator.
$\mathbf{when}(g_1 \& P \parallel g_2 \& P \parallel G) = \mathbf{when}((g_1 + g_2) \& P \parallel G)$
(2) The effect of **true** *guard is equivalent to the guard* **I**.
$\mathbf{when}(\mathbf{true} \& P \parallel G) = \mathbf{when}(\mathbf{I} \& P \parallel G)$
(3) The successive program of the **when** *construct can be distributed to every branch of the* **when** *construct.*
$\mathbf{when}(g_1 \& P_1 [\!] ... [\!] g_n \& P_n); Q = \mathbf{when}(g_1 \& (P_1; Q) [\!] ... [\!] g_n \& (P_n; Q))$
(4) The previous assignment can be distributed to every branch of the **when** *construct.*

$$(x := e); \mathbf{when}(g_1 \& P_1 [\!] ... [\!] g_n \& P_n) = \mathbf{when}(g_1[e/x] \& (x := e; P_1) [\!] ... [\!] g_n[e/x] \& (x := e; P_n))$$

(5) The branches with the same guard can be combined with nondeterministic choice.
$\mathbf{when}(g \& P \parallel g \& Q \parallel G) = \mathbf{when}(g \& (P \sqcap Q) \parallel G)$
(6) A branch with conditional choice can be divided into two branches.

$$\mathbf{when}(g \& (P \lhd b \rhd Q) \parallel G) = \mathbf{when}((b \cdot g) \& P \parallel (\neg b \cdot g) \& Q \parallel G)$$

Until Statement.
The statement R **until** g specifies a hybrid program where the change of the continuous variables is governed by the hybrid relation R until the guard condition g is triggered. It is suitable to specify the behaviour of the control plant which evolves accordingly until receiving control signals from the controlling device.

Definition 10.
Let $R(v, \dot{v})$ be a hybrid relation specifying the dynamics of the continuous variable v. Let g be a guard. Assume that all the signals of g are included in the alphabet of R, then

$$\alpha(R \ \mathbf{until} \ g) =_{df} \alpha R$$

and the behaviour of R **until** *g is described by*

$$R \ \mathbf{until} \ g =_{df} \mathbf{H}\left(\begin{pmatrix} st' = stable \ \wedge \ II_B \ \wedge \ R \wedge \\ \mathbf{time-passing} \ \wedge \ g.\mathbf{inactive} \end{pmatrix} \vee \begin{pmatrix} st' = term \ \wedge \ II_{PVar \cup B} \ \wedge \ \mathbf{update}(pos, g) \ \wedge \ R \wedge \\ \mathbf{time-passing} \ \wedge \ g.\mathbf{triggered} \end{pmatrix}\right)$$

where $B =_{df} \{s.\mathbf{clock} \mid s \in \mathbf{OutSignal}\}$.

The R **until** g statement will keep stable when the guard g is inactive during the execution interval where the continuous variables evolve as the hybrid relation R specifies. It will terminate when the guard g is triggered.

Theorem 11.
The **until** *constructor is monotonic with respect to its hybrid relation component.*

If $R1 \sqsupseteq R2$ *then* $(R1$ **until** $g) \sqsupseteq (R2$ **until** $g)$.

Example 5 (Bouncing ball). Consider the bouncing ball system. Let q be a continuous variable indicating the distance between the ball and the floor. The dynamics of the ball in its falling phase can be specified as follows.

$$Fall =_{df} ((q \geq 0 \wedge \ddot{q} = -g)\ \mathbf{init}\ \dot{q} = 0)\ \mathbf{until}\ q = 0$$

where g is the acceleration imposed by gravity.
When the ball hits the ground at time τ, its velocity $\dot{q}$ will change to $-r\dot{q}(\tau)$ instantaneously where r is a restitution coefficient ranging over $(0, 1]$. In order to set the initial value of $\dot{q}$ for the bouncing-back phase, we use the sampling assignment $x \leftarrow \dot{q}$ to copy the value of $\dot{q}(\tau)$ to a discrete variable x after $Fall$.

The dynamics of the ball in its bouncing-back phase can be specified as follows.

$$Bounce =_{df} ((q \geq 0 \wedge \ddot{q} = -g)\ \mathbf{init}\ \dot{q} = -rx)\ \mathbf{until}\ \dot{q} = 0$$

In summary, the bouncing ball system with the initial height $h_0 > 0$ from the ground can be specified as

$$BBall =_{df} (q \leftarrow h_0); (Fall; (x \leftarrow \dot{q}); Bounce)^*.$$

Note that when $r < 1$, the initial position of the ball in the falling phase is decreased for each iteration. It results to **Zeno behaviour** when the execution time is large enough and the time cost for each iteration becomes significantly close to 0. In this case, the process can perform infinite transitions within a very small time interval, which is considered as **chaos** in our model. To avoid the Zeno behaviour, we let the system stop bouncing when the speed of the ball is smaller than a small enough value δ.

$$Non\_Zeno\_BBall =_{df} (q \leftarrow h_0); (Fall; (x \leftarrow \dot{q}); (Bounce \lhd x \geq \delta \rhd \mathbf{idle}))^* \quad \square$$

Signal Hiding.
Let P be a program, and s an output signal of P. The signal hiding operator $\mathbf{signal}s \bullet P$ makes the signal s a bounded signal name of P which cannot be observed by P's environment. Through signal hiding operator, the scope of a signal can be set by the designers so that only the parallel components can react to a signal. It is helpful when modeling a distributed hybrid system where each component has limited communication capacity.

Definition 11 (Signal Hiding).
The signal hiding statement $\mathbf{signal}\ s \bullet P$ *behaves like* P *except that* s *becomes invisible to its environment.*

$\mathbf{signal}\ s \bullet P\ =_{df}\ \exists s.\mathbf{clock}' \bullet P[\epsilon/s.\mathbf{clock}]$

with $\alpha(\mathbf{signal}\ s \bullet P) =_{df} \alpha(P) \setminus \{s.\mathbf{clock}, s.\mathbf{clock}'\}$

where ϵ *denotes the empty sequence.*

Timer Declarations.
Let P be a hybrid program, and c a timer of P. The timer declaration operator $\mathbf{timer}\ c \bullet P$ declares P as the region of permitted use of timer c. A timer c is a special continuous variable with its derivation $\dot{c} = 1$.

Definition 12 (Timer declaration).
The timer declaration statement set c *to be a local timer of* P *starting from* 0.

$$\mathbf{timer}\ c \bullet P\ =_{df}\ \exists c \bullet P[0/c[t]]$$

with $\alpha(\mathbf{timer}\ c \bullet P)\ =_{df}\ \alpha(P) \setminus \{c\}$

The timer declaration operator facilitates the modeling of time-out mechanism which is one of the common reaction mechanisms in hybrid systems.

Example 6. Consider a control requirement that a program will execute P when a signal s is received within 3 s, otherwise, it will execute Q. This requirement can be specified with the time-out mechanism as follows.

$$\mathbf{timer}\ c \bullet \mathbf{when}((c < 3) \cdot s\ \&\ P \parallel (c \geq 3)\ \&\ Q) \qquad \square$$

Theorem 12.
The **delay** *command can be rewritten as the following time-out form.*

$$\mathbf{delay}(\delta)\ =\ \mathbf{timer}\ c \bullet \mathbf{idle\ until}\ (c \geq \delta)$$

Recursion.
Based on the conclusion in Sect. 3.2 and the combinators defined in above subsections, the healthy hybrid programs form a complete lattice L and is closed w.r.t. all above combinators. In this sense, we can obtain the semantics of recursive programs in this domain.

Definition 13 (Recursion).
A recursive program is defined as the weakest fixed point, denoted as $\mu X.F(X)$*, of the equation* $X\ =\ F(X)$ *in the complete lattice* L.

The notation $\nu X.F(X)$ is used to stand for the strongest fixed point of the above equation. The fixed points μF and νF are subject to the following laws:

Theorem 13.

(1) $Y \sqsupseteq \mu X.F(X)$ *whenever* $Y\ \sqsupseteq\ F(Y)$
(2) $\nu X.F(X)\ \sqsupseteq\ Y$ *whenever* $F(Y)\ \sqsupseteq\ Y$
(3) If $F(X) \sqsupseteq G(X)$ *for all* X*, then* $\mu X.F(X) \sqsupseteq \mu X.G(X)$ *and* $\nu X.F(X) \sqsupseteq \nu X.G(X)$.

For simplicity we will use the notation P^* to stand for the recursive program $\mu X.(P; X)$.

Theorem 14.
If $P \sqsupseteq Q$ *then* $P^* \sqsupseteq Q^*$

Following the concept of *approximation chain* explored in [HH98], we are going to show that under some conditions the strongest and weakest fixed points of the equation $X = P; X$ are in fact the same.

Theorem 15.
If there exists $l > 0$ *such that* $P[term, term/st, st'] \sqsupseteq (t' - t) \geq l$, *then*
(1) $\mu X.(P; X) = \nu X.(P; X)$
(2) $P^* \sqsupseteq S$ *whenever* $(P; S) \sqsupseteq S$

This theorem provides us a verification approach for the iterative program in which each iteration takes some time to finish. In other terms, it does not consider the programs with Zeno behaviours. According to this theorem, to prove that the non-Zeno iterative program satisfy a given specification, it is sufficient for us to prove its single iteration does not violate the specification.
For the systems that contains Zeno behaviour, we need to change the specification to make an approximation for avoiding the Zeno behaviour by setting a lower bound to the interval for each iteration. It can be reviewed in Example 5.

4 Conclusion

This paper proposes a hybrid modelling language, where the discrete transitions are modelled by assignment and output as zero time actions, while the continuous transitions of physical world are described by differential equations and synchronous constructs. We adopt a signal-based interaction mechanism to synchronise the activities of control programs with physical devices. A rich set of guard compositions allows us to construct sophisticated firing conditions of the transition of physical devices.
Compared with hybrid automata and HCSP, our language enriches the interaction mechanisms between processes via supporting asynchronous reactions to more complicated guards allowing combinations of testing and signals inspired by *Esterel* language. Besides, our language is equipped with true concurrency semantics of parallel composition and the communications between components are not restricted to fixed channels. In our language, the physical state of the system can be observed by all control programs and the signals can be exchanged between them with engineered protocols. It can specify the coordination control patterns in many modern control scenarios in which a set of physical objects are controlled by a network of control components.
In the future we plan to work on a proof system for hybrid program based on the algebraic laws obtained from the UTP semantics of the hybrid language. Besides the conventional sequential combinators, the proof system will focus on verification of parallel programs. We also intend to carry out non-trivial case studies with multiple physical objects and network of control components using the hybrid language and the proof system.

References

[dSS00] van der Schaft, A.J., Schumacher, J.M.: An Introduction to Hybrid Dynamical Systems. Springer, Verlag (2000)

[Bra95] Branicky, M.S.: Studies in hybrid systems: modeling, analysis, and control. Ph.D. Thesis, EECS Department, Massachusetts Institute of Technology (1995)

[ACH93] Alur, R., Courcoubetis, C., Henzinger, T.A., Ho, P.-H.: Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems. In: Grossman, R.L., Nerode, A., Ravn, A.P., Rischel, H. (eds.) HS 1991-1992. LNCS, vol. 736, pp. 209–229. Springer, Heidelberg (1993). doi:10.1007/3-540-57318-6_30

[Ben98] Benveniste, A.: Compositional and uniform modelling of hybrid systems. IEEE Trans. Autom. Control **43**(4), 579–584 (1998)

[BCP10] Benveniste, A., Cailland, B., Pouzet, M.: The fundamentals of hybrid system modelers. In: CDC, pp. 4180–4185. IEEE (2010)

[BG92] Berry, G., Gonthier, G.: The esterel synchronous programming language: design, semantics and implementation. Sci. Comput. Program. **19**(2), 87–152 (1992)

[Ber96] Berry, G.: Constructive semantics of Esterel: from theory to practice (abstract). In: Wirsing, M., Nivat, M. (eds.) AMAST 1996. LNCS, vol. 1101, pp. 225–225. Springer, Heidelberg (1996). doi:10.1007/BFb0014318

[CPP06] Carloni, L.P., Passerone, R., Pinto, A.: Languages and tools for hybrid systems design. Found. Trends Electron. Des. Autom. **1**(1/2), 1–193 (2006)

[Des96] Deshpande, A., Göllü, A., Varaiya, P.: SHIFT: a formalism and a programming language for dynamic networks of hybrid automata. In: Antsaklis, P., Kohn, W., Nerode, A., Sastry, S. (eds.) HS 1996. LNCS, vol. 1273, pp. 113–133. Springer, Heidelberg (1997). doi:10.1007/BFb0031558

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

[He94] Jifeng, H.: From CSP to hybrid systems. In: Roscoe, A.W. (ed.) a classical mind: essays in honour of C.A.R. Hoare, pp. 171–189 (1994)

[He03] Jifeng, H.: A clock-based framework for constructions of hybrid systems. In: The Proceedings of ICTAC 2013 (2013)

[Hen96] Henzinger, T.A.: The theory of hybrid automata. In: LICS, pp. 278–292. IEEE Computer Society (1996)

[Hoa85] Hoare, C.A.R.: Communicating Sequential Processes. Prentice Hall, Upper Saddle River (1985)

[HH98] Hoare, C.A.R., Jifeng, H.: Unifying Theories of Programming. Prentice Hall, Englewood Cliffs (1998)

[Koh88] Kohn, W.: A declarative theory for rational controllers. In: Proceedings of 27th CDC, pp. 130–136 (1988)

[Kra06] Kratz, F., Sokolsky, O., Pappas, G.J., Lee, I.: R-Charon, a modeling language for reconfigurable hybrid systems. In: Hespanha, J.P., Tiwari, A. (eds.) HSCC 2006. LNCS, vol. 3927, pp. 392–406. Springer, Heidelberg (2006). doi:10.1007/11730637_30

[MMP91] Maler, O., Manna, Z., Pnueli, A.: Prom timed to hybrid systems. In: Bakker, J.W., Huizing, C., Roever, W.P., Rozenberg, G. (eds.) REX 1991. LNCS, vol. 600, pp. 447–484. Springer, Heidelberg (1992). doi:10.1007/BFb0032003

[Mil99] Milner, R.: Communicating and Mobile Systems: the π-calculus. Cambridge University Press, New York (1999)

[Pla08] Platzer, A.: Differential dynamic logic: automated theorem proving for hybrid systems. Ph.D. thesis, Department of Computing Science, University of Oldenburg (2008)

[Pla10] Platzer, A.: Logical analysis of hybrid systems. In: Kutrib, M., Moreira, N., Reis, R. (eds.) DCFS 2012. LNCS, vol. 7386, pp. 43–49. Springer, Heidelberg (2012). doi:10.1007/978-3-642-31623-4_3

[RRS03] Ronkko, M., Ravn, A.P., Sere, K.: Hybrid action systems. Theoret. Comput. Sci. **290**(1), 937–973 (2003)

[RS03] Rounds, W.C., Song, H.: The Ö-calculus: a language for distributed control of reconfigurable embedded systems. In: Maler, O., Pnueli, A. (eds.) HSCC 2003. LNCS, vol. 2623, pp. 435–449. Springer, Heidelberg (2003). doi:10.1007/3-540-36580-X_32

[Sim] Simulink. www.mathworks.com/products/simulink/

[Tar41] Tarski, A.: On the calculus od relations. J. Symbolic Logic **6**(3), 73–89 (1941)

[Tar55] Tarski, A.: A lattice-theoretical fixpoint theorem and its applications. Pac. J. Math. **5**, 285–309 (1955)

[Tav87] Tavermini, L.: Differential automata and their discrete simulations. Non-Linear Anal. **11**(6), 665–683 (1987)

[ZH04] Chen, Z.C., Hansen, M.R.: Duration Calculus: A Formal Approach to Real-time Systems. Springer, Heidelberg (2004)

[ZWR96] Chaochen, Z., Ji, W., Ravn, A.P.: A formal description of hybrid systems. In: Alur, R., Henzinger, T.A., Sontag, E.D. (eds.) HS 1995. LNCS, vol. 1066, pp. 511–530. Springer, Heidelberg (1996). doi:10.1007/BFb0020972