Denotational and Algebraic Semantics for Cyber-physical Systems

Ran Li[1], Huibiao Zhu[1], Richard Banach[2]
[1]Shanghai Key Laboratory of Trustworthy Computing,
East China Normal University, Shanghai, China
[2] School of Computer Science, University of Manchester,
Oxford Road, Manchester M13 9PL, UK

***Abstract*—The cyber-physical system (CPS) is a dynamic system that contains both continuous and discrete behaviors. It has a wide range of applications in fields such as healthcare equipment, intelligent traffic control and environmental monitoring. However, the combination of continuous physical behavior and discrete control behavior may complicate the design of systems further. It is of great necessity to give an explicit formal language and its semantics for CPS. In this paper, we elaborate the modeling language for CPS based on our previous work. This language supports shared variables to model the interaction between the physical and the cyber. Additionally, we give it denotational semantics and algebraic semantics, especially focus on the continuous behavior and its composition with the discrete behavior. Throughout this paper, we also present some examples to illustrate the feasibility of the language and its semantics intuitively.**

***Index Terms*—Cyber-physical system (CPS), Unifying Theories of Programming (UTP), Denotational semantics, Algebraic semantics**

I. Introduction

The cyber-physical system (CPS) is a multidimensional complex system that integrates discrete computer control behavior and continuous physical behavior. By deeply embedding computing and communication into the physical processes, computer programs can monitor and control the physical processes. Meanwhile, the physical processes can also affect computations. CPS has an important and wide application prospect, and it has been widely applied in many fields, such as healthcare equipment, intelligent traffic control and environmental monitoring.

In addition, CPS has been an active research area for many years and some related works of formal methods have been carried out. Bu et al. explored online verification of CPS and modeling-verification-fixing framework of event-driven IoT system from bounded reachability analysis of linear hybrid automata [1]. Wang et al. proposed hyper probabilistic signal temporal logic to verify probabilistic hyperproperties for CPS [2]. Banach et al. extended Event-B to Hybrid Event-B that includes continuous behavior and discrete transitions [3].

There are also some specification languages developed for CPS as well. Inspired by the work in [4], a language to describe hybrid systems called Hybrid CSP which based on communication mechanism has been developed by Zhou et al. in [5], and Liu et al. presented a calculus for it in [6]. Ronkko et al. investigated the use of action systems with differential actions and extended hybrid action systems for hybrid systems [7]. He et al. extended the guarded command language and then presented a hybrid relational modeling language and its semantics [8].

In our paper, we elaborate the modeling language based on our previous work [9], and detail the continuous behavior according to the different situations where various guards can be triggered. It supports shared variables to model the interaction between the physical and the cyber. Our parallel mechanism, which is different from the communication mechanism of [5], [8], is based on shared variables. Besides, we give the denotational semantics and algebraic semantics of this language through the Unifying Theories of Programming (UTP) approach [10]. For the feature of shared variables in our language, two merge functions are introduced to describe the denotational semantics. With our algebraic laws, every program can be transformed into the form of guarded choice and the parallel programs with discrete and continuous behaviors can be sequentialized consequently. We also give some examples to demonstrate the usage of our language and its semantics as well.

As described in Hoare and He's UTP [10], three different mathematical models are often used to represent a theory of programming, namely, the operational, the denotational, and the algebraic approaches [11]–[13]. Each of these representations has its distinctive advantages for theories of programming. The algebraic semantics is well suited in symbolic calculation of parameters and structures of an optimal design. The denotational semantics is concerned with the mathematical objects (i.e., domains) that represent what the programs do.

The rest of the paper is organized as follows. In Sec. II, we propose the modeling language and introduce five types of guarded choices. Moreover, a Battery Management System (BMS) is shown as an example to exhibit the usage of our language. In Sec. III, we study the denotational semantics of our language. We first give the semantic model and healthiness conditions. Then, we investigate the denotational semantics of the language. In Sec. IV, we explore the algebraic semantics, including the algebraic laws for the basic statements and the parallel composition. Sect. V summarizes the paper and presents some future works.

TABLE I
SYNTAX OF CPS

Process	$P, Q ::= Db$	(Discrete Behavior)
	$\mid Cb$	(Continuous Behavior)
	$\mid P; Q$	(Sequential Composition)
	$\mid$ **if** b **then** P **else** Q	(Conditional Construct)
	$\mid$ **while** b **do** P	(Iteration Construct)
	$\mid P \parallel Q$	(Parallel Composition)
Discrete Behavior	$Db ::= x := e \mid @gd$	
Continuous Behavior	$Cb ::= R(v, \dot{v})$ **until** g	
Guard Condition	$g ::= gd \mid gc \mid gd \vee gc \mid gd \wedge gc$	
Discrete Guard	$gd ::= true \mid x = e \mid x < e \mid x > e \mid gd \vee gd \mid gd \wedge gd \mid \neg gd$	
Continuous Guard	$gc ::= true \mid v = e \mid v < e \mid v > e \mid gc \vee gc \mid gc \wedge gc \mid \neg gc$	

II. SYNTAX

In this section, we give the syntax of the language to model cyber-physical systems. Then, we introduce five types of guarded choices defined to support our algebraic expansion laws. Finally, we show an example of a Battery Management System to demonstrate the syntax of our language.

A. Syntax of CPS

As shown in Table I, we follow the syntax proposed in our previous work [9]. Here, x is a discrete variable, e is a discrete or continuous expression, v is a continuous variable and b is a boolean condition.

- $x := e$ is a discrete variable assignment. Through this assignment, the expression e is evaluated and the value gained is assigned to the variable x. The assignment is atomic, which is executed at once and other actions cannot interrupt its execution.
- $@gd$ is a discrete event guard. It is triggered when the discrete guard condition gd is fulfilled. Otherwise, it waits. Further, we assume that the shortest moment of time is one second (i.e., one time unit) in the discrete models.
- $R(v, \dot{v})$ **until** g is the syntax of describing continuous behavior in our language. $R(v, \dot{v})$ is a differential relation defining the dynamics of the continuous variable v. The evolution of v will follow the differential relation until the guard condition g is triggered.

For guard condition g, we consider two types of guard conditions and the mixed guard is permitted. gd represents guard conditions related only to discrete variables and gc contains guard conditions determined by continuous variables.

Then, a process can be comprised of the above commands. Moreover, this language supports various compositions and constructs.

- $P; Q$ is sequential composition. The process P is executed first and then Q is executed if P terminates successfully.
- **if** b **then** P **else** Q is a conditional construct. If b is true, then the process P is executed. Otherwise, Q is executed.
- **while** b **do** P is an iteration construct. The process P is executed repeatedly until the boolean condition b is no longer satisfied.
- $P \parallel Q$ is parallel composition. It indicates P executes in parallel with Q. The parallel mechanism is based on shared variables.

B. Guarded Choice

We introduce three kinds of guarded components and five types of guarded choices in order to support the algebraic parallel expansion laws in Sec. IV.

hP is a guarded component if h is $b\&@(x := e)$, $@(gd)$ or Cb^1, where b is a boolean condition and Cb^1 represents the continuous behavior performs for one second at most. $b\&@(x := e)$ is an assignment guarded component, $@(gd)$ is an event guarded component and Cb^1 is a continuous behavior guarded component.

Here, we describe Cb^1 informally and the formal definition is presented in Sec. III. If the continuous behavior Cb performs less than or equal to one second and then terminates, the behavior of Cb^1 is the same as Cb. Otherwise, we split Cb into several parts, and the behavior of Cb is the sequential composition of these parts. The introduction of Cb^1 is due to the shortest moment of time in the discrete models.

$[]\{h_1P_1, ..., h_nP_n\}$ is a guarded choice if every element in $\{h_1P_1, ..., h_nP_n\}$ is a guarded component. An assignment is instantaneous, whereas a continuous behavior is not. If an assignment guard component and a continuous behavior guard component appear in the same set of a guarded choice, Cb^1 will never have a chance to be scheduled. Thus, there is no Assignment & Cb Hybrid Guarded Choice. In our language, we present five types of guarded choices. The first three are composed of a set of one type of guard components and the last two are hybrid guarded choices.

- **Assignment Guarded Choice:**
$[]_{i\in I}\{b_i\&@(x_i := e_i)P_i\}$: It is composed of several assignment guarded components. If b_i is satisfied, $@(x_i := e_i)$ will be executed and then the corresponding program P_i will be performed. In addition, the boolean conditions $\{b_1...., b_n\}$ of the assignment guard components should satisfy $\vee_i b_i = true$.

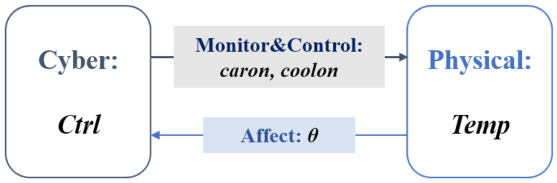

Fig. 1. A BMS Example

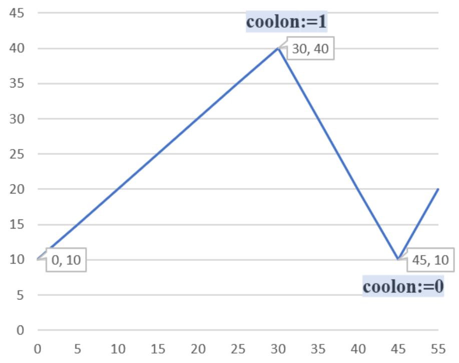

Fig. 2. Temperature Evolution

- **Event Guarded Choice:**
  $[]_{i\in I}\{@(gd_i)P_i\}$: It is composed of event guarded components and it waits for any one of the guards to be triggered. If $@(gd_i)$ is triggered, the following program P_i will be executed.
- **Cb Guarded Choice:**
  $[]\{Cb^1P\}$: It includes continuous behavior guarded component. Cb^1 will be executed and its subsequent behavior is described as the program P.
- **Assign&Event Hybrid Guarded Choice:**
  $[]_{i\in I}\{b_i\&@(x_i := e_i)P_i\}[][]_{j\in J}\{@(gd_j)Q_j\}$: It contains assignment guarded components and event guarded components. If b_i is satisfied, $@(x_i := e_i)$ can be selected to execute and then P_i will be executed. At the same time, the system is waiting for the event guard $@(gd_j)$ to be triggered. If $@(gd_j)$ can be triggered at present, the following program Q_j will be executed.
- **Event&Cb Hybrid Guarded Choice:**
  $[]_{i\in I}\{@(gd_i)P_i\}[][]\{Cb^1Q\}$: It is comprised of event guarded components and continuous behavior guarded component. The event guard $@(gd_i)$ is waiting to be triggered, and if it can be triggered at present, P_i will be executed. In the meanwhile, the continuous behavior is performing.

Based our algebraic laws proposed in Sec. IV, we can transform every program of our language into the guarded choice form, and the parallel programs with discrete and continuous behaviors can be sequentialized consequently.

C. Illustrative Example

Example 1. We now give an example of a Battery Management System (BMS) to illustrate our language. BMS is an important component of the electric vehicle power battery system and it takes charge of heat management. For simplicity, we assume that the battery works properly when the temperature is between $T_{safemin}$ and $T_{safemax}$. When the temperature is higher than $T_{safemax}$, BMS will cool the battery. Further, BMS will stop cooling if the temperature is lower than $T_{safemin}$. T_{MIN} and T_{MAX} are the low and high temperature limits, we assume the battery is out of action if the temperature is beyond this range.

Then, we model this simplified example with the above language. As shown in Fig. 1, $Ctrl$ is a discrete program defined to monitor the temperature of the battery and cool the battery through the discrete variables $caron$ and $coolon$. $Temp$ contains continuous behaviors of rising temperature when the car is moving and lowing temperature when the BMS cools the battery. This physical process can affect the cyber program through the continuous variable θ.

$$\begin{array}{ll} BMS =_{df} & caron := 1;\ DT := 0;\ coolon := 0;\ \theta_m := \theta; \\ & Ctrl \parallel Temp; \end{array}$$

$$Ctrl =_{df}\ \textbf{while}\ true\ \textbf{do}$$

$$\left\{\begin{array}{l} @(caron = 1);\ \dot{t} = 1\ \textbf{until}\ t \geqslant DT + 1;\ \theta_m := \theta; \\ \textbf{if}(\theta_m \geqslant T_{safemax})\textbf{then}\ coolon := 1;\ \textbf{else}\ coolon := coolon; \\ \textbf{if}(\theta_m \leqslant T_{safemin})\textbf{then}\ coolon := 0;\ \textbf{else}\ coolon := coolon; \\ DT := t; \end{array}\right\}$$

$$Temp =_{df}\ \textbf{while}\ true\ \textbf{do}$$

$$\left\{\begin{array}{l} \textbf{if}(coolon == 0) \\ \textbf{then}\ \dot{\theta} = 1\ \textbf{until}(\theta > T_{MAX} \vee coolon = 1 \vee caron = 0); \\ \textbf{else}\ \dot{\theta} = -2\ \textbf{until}(\theta < T_{MIN} \vee coolon = 0 \vee caron = 0); \end{array}\right\}$$

Here, $caron = 1$ represents the car is moving and $coolon = 1$ indicates the battery starts to cool down. θ_m, the measured temperature, is a discrete variable assigned by a continuous variable once a second. In our model, we assume $T_{MIN} = 0$, $T_{MAX} = 100$, $T_{safemin} = 10$ and $T_{safemax} = 40$. The initial temperature is 10 and the temperature evolution is shown in Fig. 2.

III. Denotational Semantics

In this section, we investigate the denotational semantics of our language based on the UTP approach. We first give the semantic model and healthiness conditions. On this basis, we present denotational semantics and the example of BMS is discussed again in this section.

A. Semantic Model

We take a tuple $(time, time', st, st', tr, tr')$ to represent the observation of a cyber-physical system program. Here,

- $time$ and $time'$ represent the start point time and the end point time of an observation time interval. We use δ to represent the length of the time interval, i.e., $\delta(time)_{df} = (time' - time)$.
- st and st' indicate the state of the program at the start point and the end point respectively. In our language, we consider that the program may have three types of state: $term$, $wait$ and div.
  - $term$: If $st = term$, it implies that the previous program has terminated and the current program can begin to be executed. If $st' = term$, it indicates

that the current program has been terminated and the following program can be executed.
  - $wait$: If $st = wait$, it means the current program cannot be activated because the previous program has not terminated. If $st' = wait$, it represents the current program is waiting so that the next program cannot be activated.
  - div: If $st = div$, it shows that the previous program reaches div state (i.e., divergent state) and never terminates so that the current can never be executed. If $st' = div$, it means that the current program diverges and the following program can never be executed.
- tr and tr' are the traces at the start point and the end point respectively.
  We use a series of snapshots that record a sequence of actions to describe the behavior of a program. A snapshot in our semantic model can be expressed as a triple (t, σ, μ).
  - t: It records the time at which the action occurs.
  - σ: We use σ to record the states of data (i.e., discrete variables) contributed by the program itself or its environment during the program runtime.
  - μ: We introduce it to indicate whether the action is done by the program itself or by the environment. If the program itself executes this action, we set μ to 1. Otherwise, μ equals 0 if the action is done due to the behaviors of the environment.

  We use the projection function $\pi_i (i = 1, 2, 3)$ to gain the ith component of a snapshot. Further, we mark the last snapshot with the symbol of $last(tr)$. $tr_2 - tr_1$ denotes the rest of tr_2 after removing the snapshots of tr_1. $tr_1 {}^\frown tr_2$ denotes the concatenation of the trace tr_1 and tr_2.

$$\pi_1((t,\sigma,\mu)) =_{df} t, \quad \pi_2((t,\sigma,\mu)) =_{df} \sigma, \quad \pi_3((t,\sigma,\mu)) =_{df} \mu$$

In addition, the programs need to satisfy some healthiness conditions and we consider three conditions in our model. Here, we assume that a program is healthy if it caters for these healthiness conditions, that is, healthiness conditions can be considered as properties of programs that need to be satisfied.

$$\begin{aligned} &H1 : P = P \wedge (tr \le tr') \wedge (time \le time') \\ &H2 : P = \Pi \lhd st = wait \rhd P \\ &H3 : P = \perp \lhd st = div \rhd P \end{aligned}$$

The healthiness condition $H1$ requires that the trace and the time of a program can only grow but cannot shorten. The healthiness condition $H2$ describes the following two requirements. If $st = wait$, it means the current program is waiting for the termination of the previous program so that all variables are not changed. If $st \neq wait$, it implies the current program can be executed. The healthiness condition $H3$ indicates that the behavior of P is totally unpredictable if its state is div. Here, Π, $P \lhd b \rhd Q$ and $\perp$ are defined below.

$$\begin{aligned} &\Pi =_{df} (st = st') \wedge (tr = tr') \wedge (time = time') \\ &P \lhd b \rhd Q =_{df} (b \wedge P) \vee (\neg b \wedge Q) \\ &\perp =_{df} true \end{aligned}$$

Next, we define $H(X)$ function used to define the denotational semantics for our language. The parameter X, which is different from the syntax of a program, is the description of a program containing elements from the tuple $(time, time', st, st', tr, tr')$ based on the semantic model. The following definition implies $H(X)$ caters for the healthiness conditions $H1$, $H2$ and $H3$.

$$H(X) = df \ \perp \lhd st = div \rhd \left(\begin{array}{c} \Pi \lhd st = wait \rhd \\ \left(\begin{array}{c} X \wedge (tr \le tr') \\ \wedge (time \le time') \end{array} \right) \end{array} \right)$$

For simplicity of describing the continuous behaviors, we define $R \wedge_H H(X)$ based on $H(X)$, where R is a differential relation.

$$R \wedge_H H(X) =_{df} H(R \wedge X)$$

B. Semantics of Basic Statements

In this subsection, we give the denotational semantics of basic statements. In our paper, we present the denotational semantic of P by the notation of $beh(P)$ to differentiate the syntax and the semantics of P.

1) $x := e$:

Since assignment operation is an instantaneous action, it takes no time and terminates at once without waiting. It assigns the value of e to the discrete variable x. Here, $\sigma[e/x]$ is the same as σ except the value of variable x is now associated with the value e. When the data state after the assignment is different from the current state of $last(tr)$, we update the trace tr' by adding the assignment result to the end. We require $\pi_3(envtr) \in 0^*$ and it means that the environment (i.e., other programs given by parallel composition) can do actions before the assignment.

$$beh(x := e) =_{df} H \left(\begin{array}{l} st' = term \wedge \delta(time) = 0 \wedge \\ \left(\begin{array}{l} tr' = tr \\ \lhd \pi_2(last(tr)) = \sigma[e/x] \rhd \\ tr' = tr {}^\frown envtr {}^\frown \langle (t, \sigma[e/x], 1) \rangle \end{array} \right) \end{array} \right)$$

2) @(gd):

$@gd$ is a discrete event guard that can be triggered when the discrete guard condition gd is fulfilled. We think about the following two scenarios.

- Scenario 1: Consider the program P below. $@(x > 1)$ can be triggered due to P's own assignment $x := 2$.

$$P \ = \ x := 0; x = 2; @(x > 1)$$

- Scenario 2: Consider the parallel program $P \parallel Q$ below. $@(x > 1)$ in P cannot be triggered by itself and it will wait for the environment's actions to trigger this guard, i.e., $@(x > 1)$ in P can be triggered by Q's assignment $x := 3$.

$$P \ = \ x := 0; x = 1; @(x > 1), \qquad Q \ = \ x := 3$$

Based on the above analysis, we can conclude that $@gd$ can be triggered by itself or by the environment. We give the denotational semantic as below. Here, ; is a sequence operator and its denotational semantic is presented in Sec. III-B5.

$$beh(@gd) =_{df} selftrig(gd) \vee (await(gd); envtrig(gd))$$

$selftrig(gd)$ implies that the event guard is triggered by the program's own action and its specific definition is shown below. Since the discrete event guard is instantaneous, it terminates immediately and takes no time. Besides, $@gd$ cannnot change the data states of the program and the trace cannot change consequently. $gd(\sigma)$ means gd is true in the data state σ and $gd(\pi_2(last(tr)))$ indicates that gd can be triggered at the beginning due to its own action.

$$selftrig(gd) =_{df} H\left(\begin{array}{l} st' = term \wedge \delta(time) = 0 \wedge \\ tr' = tr \wedge gd(\pi_2(last(tr))) \end{array}\right)$$

$await(gd); envtrig(gd)$ represents that the event guard gd cannot be triggered by itself and it waits for actions from the environment. We give the definitions of $await(gd)$ and $envtrig(gd)$ as below.

$$await(gd) =_{df} H\left(\begin{array}{l} st' \neq div \wedge \neg gd(\pi_2(last(tr))) \wedge \\ \forall s \in (tr' - tr) \cdot \neg gd(\pi_2(s)) \wedge \\ \pi_3(tr' - tr) \in 0^* \end{array}\right)$$

$$envtrig(gd) =_{df} H\left(\begin{array}{l} st' = term \wedge \delta(time) = 0 \wedge \\ gd(\pi_2(last(tr))) \wedge \\ \pi_3(tr' - tr) = 0 \wedge len(tr' - tr) = 1 \end{array}\right)$$

When the guard cannot be triggered by itself, the program will wait. $await(gd)$ defines the behaviors of waiting. As it waits, the environment can do its actions and all these actions cannot trigger the guard. Here, $\pi_3(tr' - tr) \in 0^*$ means these newly added snapshots are all contributed by the environment.

$envtrig(gd)$ describes the behaviors that the environment finally triggers the guard. Similar to $selftrig(gd)$, it takes no time and terminates immediately. $\pi_3(tr' - tr) = 0$ intimates the action which triggers the guard is done by the environment. $len(tr' - tr) = 1$ emphasizes this action is the last one in the trace.

*3) $R(v, \dot{v})$ **until** g:*

We give the denotational semantics of the continuous behavior according to the types of the guard g. We divide g into four types: gd, gc, $gd \vee gc$ and $gd \wedge gc$.

- $g \equiv gd$: The continuous variable v evolves as the hybrid relation $R(v, \dot{v})$ specifies until the guard gd is triggered. Same as the discrete event guard, gd can be triggered by itself or the environment.

$$beh(R(v, \dot{v}) \textbf{ until } gd) =_{df}$$
$$\left(\begin{array}{l} R(v, \dot{v}) \wedge_H \\ (selftrig(gd) \vee (await(gd); envtrig(gd))) \end{array}\right)$$

- $g \equiv gc$: gc includes guard conditions determined by continuous variables. Similar to $R(v, \dot{v})$ **until** gd, v evolves according to the relation $R(v, \dot{v})$ until the guard gc is satisfied.

$$beh(R(v, \dot{v}) \textbf{ until } gc) =_{df}$$
$$\left(\begin{array}{l} R(v, \dot{v}) \wedge_H \\ (trig(gc) \vee (await2(gc); trig(gc))) \end{array}\right)$$

Different from gd, gc can only be triggered by itself since its value only depends on the hybrid relation $R(v, \dot{v})$. We give the definitions of $trig(gc)$ and $await2(gc)$ as below.

$$trig(gc) =_{df} H(st' = term \wedge \delta(time) = 0 \wedge gc(v(time')))$$
$$await2(gc) =_{df} H(st' \neq div \wedge \forall \tau \in [time, time') \cdot \neg gc(v(\tau)))$$

Here, $gc(v(time'))$ means that the value of v at the time $time'$ makes gc true. $\forall \tau \in [time, time') \cdot \neg gc(v(\tau))$ indicates that gc cannot be triggered at any time while it is waiting for being triggered.

- $g \equiv gd \vee gc$: It is a hybrid guard which can be triggered when gd or gc is triggered. The continuous variable v evolves until the guard gd or gc is satisfied. There are three kinds of situations where g is triggered.
  - $selftrig(gd)$: gd is triggered by itself at the beginning.
  - $trig(gc)$: gc is triggered at the beginning.
  - $(await(gd) \wedge await2(gc)); (envtrig(gd) \vee trig(gc))$: If gd and gc are not triggered at the beginning, it will wait for being triggered until one of them is triggered.

$$beh(R(v, \dot{v}) \textbf{ until } (gd \vee gc)) =_{df}$$
$$\left(R(v, \dot{v}) \wedge_H \left(\begin{array}{l} selftrig(gd) \vee trig(gc) \vee \\ \left(\begin{array}{l} (await(gd) \wedge await2(gc)); \\ (envtrig(gd) \vee trig(gc)) \end{array}\right) \end{array}\right)\right)$$

- $g \equiv gd \wedge gc$: It is a hybrid guard which can be triggered only when both gd and gc are triggered. The variable v evolves until the guard gd and gc are both satisfied. There are two kinds of situations where g is triggered.
  - $selftrig(gd) \wedge trig(gc)$: gd and gc are both triggered at the beginning.
  - $(await(gd) \vee await2(gc)); ((selftrig(gd) \vee envtrig(gd)) \wedge trig(gc))$: If gd or gc is not triggered at the beginning, it will wait for being triggered until both of them are triggered.

$$beh(R(v, \dot{v}) \textbf{ until } (gd \wedge gc)) =_{df}$$
$$\left(\begin{array}{l} R(v, \dot{v}) \wedge_H \\ \left(\begin{array}{l} (selftrig(gd) \wedge trig(gc)) \vee \\ \left(\begin{array}{l} (await(gd) \vee await2(gc)); \\ ((selftrig(gd) \vee envtrig(gd)) \wedge trig(gc)) \end{array}\right) \end{array}\right) \end{array}\right)$$

4) Condition Construct:

The denotational semantic of the condition construct is shown below. Here, $SKIP$ stands for $x := x$.

$$beh(\textbf{if } b \textbf{ then } P \textbf{ else } Q) = beh(SKIP; P) \lhd b \rhd beh(SKIP; Q)$$

5) Sequential Composition:

$P; Q$ represents executes P and Q sequentially and we first define the sequence operator ; in our semantic model as below.

- If neither of the two processes contains continuous behavior, then

$$P; Q = \exists m, s, n \cdot$$
$$P[m/time', s/st', n/tr'] \wedge Q[m/time, s/st, n/tr].$$

- If one of the two processes contains continuous behavior (i.e., it contains statements such as $R(v, \dot{v})$ **until** g), then the trace of the sequential composition stays the same as the discrete process and the value of the continuous variable v depends on the continuous process.

$$P; Q = \exists m, s \cdot P[m/time', s/st'] \wedge Q[m/time, s/st].$$

- If both of the two processes contain continuous behavior, then we similarly define the continuous variable v as $time$ and st. Here, v and v' stand for the initial value and the final value of the continuous variable respectively.

$$P;Q = \exists m,s,x\cdot$$
$$P[m/time', s/st', x/v'] \wedge Q[m/time, s/st, x/v].$$

Further, the semantic of sequential composition is given.

$$beh(P;Q) = beh(P);beh(Q)$$

6) Iteration Construct:
We define iteration construct in the same way in conventional programming languages. $\mu_{HF}F(X)$represents the weakest fixed point of the monotonic function F over the set of healthy formulae.

$$\textbf{while } b \textbf{ do } P =_{df} \mu_{HF}X \bullet \textbf{if } b \textbf{ then } (P;X) \textbf{ else } SKIP$$

C. Illustrative Example: Continuation 1

Example 2. Let us consider the programs in *Example 1* (Page 3) in Sect. II again and explore the traces of these programs. First, we give the trace of the initialization before the parallel programs as below.

$$\langle (0,\sigma_0,1),(0,\sigma_0',1),(0,\sigma_0'',1),(0,\sigma_0''',1)\rangle$$

Here, $\sigma_0 = \{caron \mapsto 1\}$, $\sigma_0' = \{caron \mapsto 1, DT \mapsto 0\}$, $\sigma_0'' = \{caron \mapsto 1, DT \mapsto 0, coolon \mapsto 0\}$, $\sigma_0''' = \{caron \mapsto 1, DT \mapsto 0, coolon \mapsto 0, \theta_m \mapsto 10\}$.
Then, we cosider the parallel programs $Ctrl||Temp$. A trace of $Ctrl$ is presented as follows.

$$\langle (1,\sigma_1,1),(1,\sigma_1',1),(2,\sigma_2,1),(2,\sigma_2',1)...(30,\sigma_{30},1),$$
$$(30,\sigma_{30}',1)(30,\sigma_{30}'',1),(31,\sigma_{31},1)(31,\sigma_{31}',1)...\rangle$$

Here,
$$\sigma_1 = \{caron \mapsto 1, DT \mapsto 0, coolon \mapsto 0, \theta_m \mapsto 11\}$$
$$\sigma_1' = \{caron \mapsto 1, DT \mapsto 1, coolon \mapsto 0, \theta_m \mapsto 11\}$$
$$\sigma_2 = \{caron \mapsto 1, DT \mapsto 1, coolon \mapsto 0, \theta_m \mapsto 12\}$$
$$\sigma_2' = \{caron \mapsto 1, DT \mapsto 2, coolon \mapsto 0, \theta_m \mapsto 12\}...$$
$$\sigma_{30} = \{caron \mapsto 1, DT \mapsto 29, coolon \mapsto 0, \theta_m \mapsto 40\}$$
$$\sigma_{30}' = \{caron \mapsto 1, DT \mapsto 29, coolon \mapsto 1, \theta_m \mapsto 40\}$$
$$\sigma_{30}'' = \{caron \mapsto 1, DT \mapsto 30, coolon \mapsto 1, \theta_m \mapsto 40\}$$
$$\sigma_{31} = \{caron \mapsto 1, DT \mapsto 30, coolon \mapsto 0, \theta_m \mapsto 38\}$$
$$\sigma_{31}' = \{caron \mapsto 1, DT \mapsto 31, coolon \mapsto 0, \theta_m \mapsto 38\}.$$

Besides, for the continuous program $Temp$, it is not necessary to record its trace. Since the trace only records discrete actions, the continuous program just provides values of continuous variables and the length of the time interval for the parallel composition.

D. Semantics of Guarded Choice

As mentioned before, we introduce five types of guarded choices. We present the denotational semantics for them.

- **Assignment Guarded Choice:**
The program $[]_{i\in I}\{b_i\&@(x_i := e_i)P_i\}$ performs one of the assignments non-deterministically and then executes the corresponding process P_i. Here, $beh(b_i\&@(x_i := e_i)) = b_i \wedge beh(x_i := e_i)$.

$$beh([]_{i\in I}\{b_i\&@(x_i := e_i)P_i\}) =_{df}$$
$$\bigvee_{i\in I} beh(b_i\&@(x_i := e_i);P_i)$$

- **Event Guarded Choice:**
The program $[]_{i\in I}\{@(gd_i)P_i\}$ waits for one of the guards to be triggered and then behaves as the following guard process. Here, gd is a compound guard and $gd = \vee_{i\in I}\ gd_i$.

$$beh([]_{i\in I}\{@(gd_i)P_i\}) =_{df}$$
$$\bigvee_{i\in I}\{(selftrig(gd_i) \vee (await(gd);trig(gd_i)));beh(P_i)\}$$

- **Cb Guarded Choice:**
The program $[]\{Cb^1P\}$ first executes the continuous behavior defined by Cb^1 and then behaves as the rest program. Here, $beh(Cb^1) = beh(Cb) \wedge (\delta(time) < 1)$.

$$beh([]\{Cb^1P\}) = beh(Cb^1;P)$$

- **Assign&Event Hybrid Guarded Choice:**
$HGC1 = []_{i\in I}\{b_i\&@(x_i := e_i)P_i\}[][]_{j\in J}\{@(gd_j)Q_j\}$
The program $HGC1$ executes the selected assignment guard $@(x_i := e_i)$ and then behaves like the corresponding process P_i. It can also start an event guarded process $@(gd_j)$ if this guard is triggered immediately and then behaves like Q_j.

$$beh(HGC1) = \begin{pmatrix} \bigvee_{i\in I} beh(b_i\&@(x_i := e_i);P_i) \\ \bigvee \\ \bigvee_{j\in J}\{(selftrig(g_j);beh(Q_j)\} \end{pmatrix}$$

- **Event&Cb Hybrid Guarded Choice:**
$HGC2 = []_{i\in I}\{@(gd_i)P_i\}[][]\{Cb^1Q\}$
The hybrid guarded choice is defined in a similar way where the event guards can only be triggered before Cb^1 begins. Therefore, the semantic of the program $HGC2$ is defined as below.

$$beh(HGC2) = \begin{pmatrix} \bigvee_{i\in I}\{selftrig(g_i);beh(P_i)\} \\ \bigvee beh(Cb^1;Q) \end{pmatrix}$$

E. Parallel Composition

We present the denotational semantic of the parallel composition as below.

$$beh(P \parallel Q) = beh(P) \parallel beh(Q),$$

where, $beh(P) \parallel beh(Q) =$

$$\begin{pmatrix} \exists st, st', time, time', tr, tr' \bullet \\ \begin{pmatrix} st_1 = st_2 = st \wedge time_1 = time_2 = time \\ \wedge tr_1 = tr_2 = tr \wedge \\ beh(P)[st/st_1, st'/st_1', time/time_1, \\ \quad time'/time_1', tr/tr_1, tr'/tr_1'] \\ \wedge beh(Q)[st/st_2, st'/st_2', time/time_2, \\ \quad time'/time_2', tr/tr_2, tr'/tr_2'] \end{pmatrix} \\ \wedge time' = max\{time_1', time_2'\} \\ \wedge mergeState(st_1', st_2') \wedge mergeTrace(tr_1', tr_2') \end{pmatrix}.$$

It requires that P and Q have the same initial values on st, $time$ and tr. $beh(P)$ and $beh(Q)$ stand for the denotational semantics of P and Q respectively. The terminal time of the parallel composition is determined by the maximum terminal time of the two parallel components. $mergeState$ and $mergeTrace$ define the terminal state and trace of the parallel composition.

If the terminal states of the parallel components P and Q are both $term$, the final state of $P \parallel Q$ is $term$. If either component stays at a $wait$ state, the state of $P \parallel Q$ is $wait$ as well. $P \parallel Q$ is divergent when either component behaves chaotically.

$$mergeState(st_1', st_2') = \left(\begin{array}{l} (st_1' = term \wedge st_2' = term \rightarrow st' = term) \wedge \\ ((st_1' = wait \wedge st_2' \neq div) \vee (st_2' = wait \wedge st_1' \neq div) \\ \rightarrow st' = wait) \wedge \\ (st_1' = div \vee st_2' = div \rightarrow st' = div) \end{array} \right)$$

The trace of the parallel composition is formulated through the interleaving of atomic actions performed by its two components. As mentioned earlier, a snapshot in the trace is a triple (t, σ, μ). The first two lines of $mergeTrace$ indicate the time and the data states recorded in the snapshots of parallel composition should be the same as that in both components. The third line implies the action of the parallel composition is done by the interleaving of the parallel components and the fourth line requires that every state can only be contributed by one of the parallel components.

$$mergeTrace(tr_1', tr_2') = \left(\begin{array}{l} (\pi_1(tr') = \pi_1(tr_1') = \pi_1(tr_2')) \wedge \\ (\pi_2(tr') = \pi_2(tr_1') = \pi_2(tr_2')) \wedge \\ (\pi_3(tr') = \pi_3(tr_1') + \pi_3(tr_2')) \wedge (2 \notin \pi_3(tr')) \end{array} \right)$$

To get a better understanding of the semantics of parallel programs, we take the BMS for instance in the next subsection.

F. Illustrative Example: Continuation 2

Example 3. We continue to explore *Example 2* (Page 6). As shown above, we have given a trace of $Ctrl$ individually. Now, we assume that there exists an radical controller $RCtrl$ and explore the situation where $Ctrl$ and $RCtrl$ run in parallel. We first give the process of $RCtrl$ as below.

$$RCtrl =_{df} \textbf{ while } true \textbf{ do} \left\{ \begin{array}{l} @(caron = 1); \\ \textbf{if}(\theta_m \geqslant T_{safemax})\textbf{then } caron := 0; \textbf{ else } caron := caron; \\ \textbf{if}(\theta_m \leqslant T_{safemin})\textbf{then } caron := 0; \textbf{ else } caron := caron; \end{array} \right\}$$

We use Fig. 3 to indicate the trace behavior of the process $Ctrl$ (and its environment, i.e., the process $RCtrl$). For the process $Ctrl$, $\bullet$ stands for its own atomic action and $\circ$ represents its environment's (i.e., $RCtrl$'s) atomic action. The vertical line shows the snapshot sequences in the traces of $Ctrl$, whereas the horizontal line denotes the time when these actions happen. Therefore, we give the traces of $Ctrl$, $RCtrl$ and $Ctrl||RCtrl$.

- $Ctrl$: All actions are done by itself except for the last one.

$$\langle (1, \sigma_1, 1), (1, \sigma_1', 1), (2, \sigma_2, 1), (2, \sigma_2', 1), ..., (30, \sigma_{30}, 1), (30, \sigma_{30}', 1)(30, \sigma_{30}'', 1), (30, \sigma_{30}'', 1), (30, \sigma_{30}''', 0) \rangle$$

- $RCtrl$: All actions are done by the environment except for the last one.

$$\langle (1, \sigma_1, 0), (1, \sigma_1', 0), (2, \sigma_2, 0), (2, \sigma_2', 0), ..., (30, \sigma_{30}, 0), (30, \sigma_{30}', 0)(30, \sigma_{30}'', 0), (30, \sigma_{30}'', 0), (30, \sigma_{30}''', 1) \rangle$$

- $Ctrl||RCtrl$:

$$\langle (1, \sigma_1, 1), (1, \sigma_1', 1), (2, \sigma_2, 1), (2, \sigma_2', 1), ..., (30, \sigma_{30}, 1), (30, \sigma_{30}', 1)(30, \sigma_{30}'', 1), (30, \sigma_{30}'', 1)(30, \sigma_{30}''', 1) \rangle$$

Here, $\sigma_{30}''' = \{caron \mapsto 0, DT \mapsto 30, coolon \mapsto 1, \theta_m \mapsto 40\}$. Other states of data agree in the states of *Example 2*.

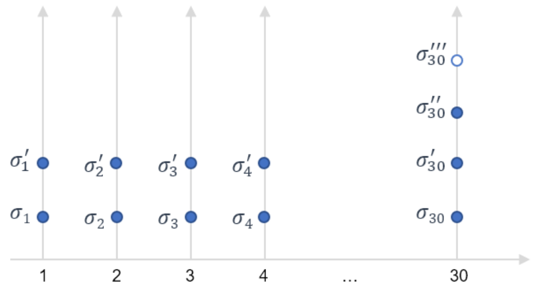

Fig. 3. Trace Behaviors of Parallel Composition

IV. Algebraic Semantics

Program properties can be expressed as algebraic laws, which can be verified using the formalized semantics. In this section, we look into the algebraic semantics and employ the example of BMS to show the usage of our algebraic laws.

A. Algebraic Laws for Basic Statements

First, we study the algebraic laws for basic statements as below. These laws imply that a program in our language can be transformed into the guarded choice form. Here, ε denotes an empty program and $(R(v, \dot{v})$ **until** $g)^1$ (i.e., Cb^1) means the continuous behavior can perform for one second at most.

- **(assign-1)** $x := e \ = [] \{true \& @(x := e)\varepsilon\}$
- **(guard-1)** $@(gd) \ = [] \{@(gd)\varepsilon\}$
- **(cb-1)** $R(v, \dot{v})$ **until** g
  $= [] \{(R(v, \dot{v}) \textbf{ until } g)^1 R(v, \dot{v}) \textbf{ until } g\}$
- **(cond-1)** **if** b **then** P **else** Q
  $= [] \{b \& @(x := x)P, \neg b \& @(x := x)Q\}$
- **(iter-1)** **while** b **do** P
  $= [] \left\{ \begin{array}{l} b \& @(x := x)(P; \textbf{while } b \textbf{ do } P), \\ \neg b \& @(x := x)\varepsilon \end{array} \right\}$
- **(seq-1)** $(P; Q); R = P; (Q; R)$
- **(seq-2)** If $P = [] \{g_1 P_1, ..., g_n P_n\}$, then
  $P; Q = [] \{g_1(P_1; Q), ..., g_n(P_n; Q)\}$.

Our denotational semantics in this paper can support the proofs of these algebraic laws and we take **(cond-1)** as an example to show that our semantics definitions are rigorous.

Proof.

$$\begin{array}{ll} beh([] \{b \& @(x := x)P, \neg b \& @(x := x)Q\}) & \\ = beh(b \& @(x := x)P) \vee beh(\neg b \& @(x := x)Q) & \\ & \{\text{Def of Guarded Choice}\} \\ = (beh(b \& @(x := x)); beh(P)) \vee (beh(\neg b \& @(x := x)); beh(Q)) & \\ & \{\text{Def of Guarded Choice}\} \\ = ((b \wedge beh(x := x)); beh(P)) \vee ((\neg b \wedge beh(x := x)); beh(Q)) & \\ & \{\text{PL}\} \\ = (b \wedge (beh(x := x); beh(P))) \vee (\neg b \wedge (beh(x := x); beh(Q))) & \\ & \{\text{PL}\} \\ = (beh(x := x); beh(P)) \lhd b \rhd (beh(x := x); beh(Q)) & \\ & \{\text{Def of SKIP}\} \\ = beh(SKIP; P) \lhd b \rhd beh(SKIP; Q) & \{\text{Def of Conditional}\} \\ = beh(\textbf{if } b \textbf{ then } P \textbf{ else } Q) & \end{array}$$

$\square$

TABLE II
COMPOSITION OF GUARD CHOICES

	Assignment	**Event**	**Cb**	**Assign&Event**	**Event&Cb**
Assignment	(par-1-1)	(par-1-2)	(par-1-3)	(par-1-4)	(par-1-5)
Event		(par-2-2)	(par-2-3)	(par-2-4)	(par-2-5)
Cb			(par-3-3)	(par-3-4)	(par-3-5)
Assign&Event				(par-4-4)	(par-4-5)
Event&Cb					(par-5-5)

B. Algebraic Laws for Parallel Composition

Parallel composition is symmetric and associative.

- **(par-1)** $P \parallel Q = Q \parallel P$
- **(par-2)** $(P \parallel Q) \parallel R = P \parallel (Q \parallel R)$

Then, we look into algebraic laws of the parallel composition for guarded choices. As mentioned previously, there are five types of guarded choices. Thus, there should be 25 expansion laws. As shown in Table II, we only need to list 15 laws because of the symmetry of the parallel composition **(par-1)**.

As presented in Table III, we only present the algebraic laws where cb guarded choice is involved due to space constraints. The rest laws concerning purely discrete behaviors agree in the parallel composition laws in [14].

- **(par-1-3)** reflects the parallel composition of assignment guarded choice and cb guarded choice. Since the assignment is an instantaneous action, it can be scheduled at once. Further, cb guarded choice is a continuous behavior that may perform for a while. For the whole parallel process, if an assignment is scheduled, the subsequent behavior is the parallel composition of the following process after this assignment and another parallel part. In general, the instantaneous assignment is considered to be executed first when we sequentialize the parallel trace of assignment and continuous behaviors.
- **(par-2-3)** exhibits the parallel composition of event guarded choice and cb guard choice. Similar to **(par-1-3)**, for the whole process, if an event guard is triggered, the subsequent behavior is the parallel composition of the rest process after this event guard and another parallel part. However, for the instantaneous action event guard, which is different from the assignment, the guard may be waiting to be triggered. Then, the continuous behavior will perform and the subsequent behavior of the whole process is the parallel composition of the rest process after this cb guarded choice and another parallel part.
- **(par-3-3)** demonstrates the condition where both of the parallel parts are cb guarded choices. The schedule rule permits that both of the cb guarded choices to perform. Since the dynamics of a continuous variable can be defined by one and only one relation in a moment, we assume that Cb_s^1 and Cb_t^1 have no continuous shared variables. $Cb_s^1|Cb_t^1$ is a conjunction of the two dynamics, i.e., the two different continuous behaviors can perform as their corresponding differential relations defined at the same time.
- **(par-3-4)** is the parallel composition of cb guarded choice and assign&event guard choice. Based on **(par-1-3)** and **(par-2-3)**, the schedule has two types. If the assignment is selected, for the whole process, the subsequent behavior is the parallel composition of the following process after this assignment and another parallel part. If the event guard is triggered, the successive behavior is composed by the rest process after the guard and another parallel part.
- **(par-1-5)** describes the parallel composition of assignment guarded choice and event&cb guarded choice. This case is expressed similarly as **(par-3-4)**.
- **(par-2-5)** is the parallel composition of event guarded choice and event&cb guarded choice. As both of the parallel parts have event guarded choices, there are three types for the guards to be triggered as listed in the first three lines of the law. Besides, the schedule rule also allows the continuous behavior to perform first and the subsequent behavior is similar to that in **(par-2-3)**.
- **(par-3-5)** illustrates the parallel composition of cb guard choices and event&cb guarded choice. Based on **(par-2-3)** and **(par-3-3)**, the subsequent behavior is the parallel composition of the following process after the event guard choices and another parallel part, or the parallel composition of the successive processes after the two cb guard choices.
- **(par-4-5)** describes the parallel composition of assign&event guard choice and event&cb guarded choice. There are three types of event guards to be triggered and the rest is the parallel composition of the following action after the event guard and another parallel part. Also, it enables the assignment to be scheduled first and the parallel composition of the following process after the assignment and another parallel part forms the subsequent behavior.
- **(par-5-5)** is the situation where both of the parallel parts are event&cb guarded choices. Similar to **(par-4-5)**, there are three types of event guards to be triggered. Besides, the schedule rule also permits the continuous behavior to be selected first if all the guards cannot be triggered at once.

TABLE III
ALGEBRAIC LAWS FOR PARALLEL COMPOSITION

(par)	P	Q	$P \parallel Q$
(par-1-3)	$[]_{i\in I}\{b_i\&@(x_i := e_i)P_i\}$	$[]\{Cb^1 S\}$	$[]_{i\in I}\{b_i\&@(x_i := e_i)(P_i \parallel Q)\}$
(par-2-3)	$[]_{i\in I}\{@(\xi_i)P_i\}$	$[]\{Cb^1 S\}$	$[]_{i\in I}\{@(\xi_i)(P_i \parallel Q)\}$ $[][]\{Cb^1(P \parallel S)\}$
(par-3-3)	$[]\{Cb^1_s S\}$	$[]\{Cb^1_t T\}$	$[]\{Cb^1_s \mid Cb^1_t(S \parallel T)\}$
(par-3-4)	$[]\{Cb^1 N\}$	$[]_{j\in J}\{b_j\&@(x_j := e_j)S_j\}$ $[][]_{k\in K}\{@(\xi_k)T_k\}$	$[]_{j\in J}\{b_j\&@(x_j := e_j)(P \parallel S_j)\}$ $[][]_{k\in K}\{@(\xi_k)(P \parallel T_k)\}$
(par-1-5)	$[]_{i\in I}\{b_i\&@(x_i := e_i)P_i\}$	$[]_{j\in J}\{@(\xi_j)S_j\}$ $[][]\{Cb^1 T\}$	$[]_{i\in I}\{b_i\&@(x_i := e_i)(P_i \parallel Q)\}$ $[][]_{j\in J}\{@(\xi_j)(P \parallel S_j)\}$
(par-2-5)	$[]_{i\in I}\{@(\xi_i)P_i\}$	$[]_{j\in J}\{@(\eta_j)S_j\}$ $[][]\{Cb^1 T\}$	$[]_{i\in I}\{@(\xi_i \wedge \neg\eta)(P_i \parallel Q)\}$ $[][]_{j\in J}\{@(\neg\xi \wedge \eta_j)(P \parallel S_j)\}$ $[][]_{i\in I\wedge j\in J}\{@(\xi_i \wedge \eta_j)(P_i \parallel S_j)\}$ $[][]\{Cb^1(P \parallel T)\}$
(par-3-5)	$[]\{Cb^1_n N\}$	$[]_{j\in J}\{@(\xi_j)S_j\}$ $[][]\{Cb^1_t T\}$	$[]_{j\in J}\{@(\xi_j)(P \parallel S_j)\}$ $[][]\{Cb^1_n \mid Cb^1_t(N \parallel T)\}$
(par-4-5)	$[]_{i\in I}\{b_i\&@(x_i := e_i)R_i\}$ $[][]_{j\in J}\{@(\xi_j)S_j\}$	$[]_{k\in K}\{@(\eta_k)T_k\}$ $[][]\{Cb^1 N\}$	$[]_{i\in I}\{b_i\&@(x_i := e_i)(R_i \parallel Q)\}$ $[][]_{j\in J}\{@(\xi_j \wedge \neg\eta)(S_j \parallel Q)\}$ $[][]_{k\in K}\{@(\neg\xi \wedge \eta_k)(P \parallel T_k)\}$ $[][]_{j\in J\wedge k\in K}\{@(\xi_j \wedge \eta_k)(S_j \parallel T_k)\}$
(par-5-5)	$[]_{i\in I}\{@(\xi_i)R_i\}$ $[][]\{Cb^1_s S\}$	$[]_{k\in K}\{@(\eta_k)T_k\}$ $[][]\{Cb^1_n N\}$	$[]_{i\in I}\{@(\xi_i \wedge \neg\eta)(R_i \parallel Q)\}$ $[][]_{k\in K}\{@(\neg\xi \wedge \eta_k)(P \parallel T_k)\}$ $[][]_{i\in I\wedge k\in K}\{@(\xi_j \wedge \eta_k)(R_i \parallel T_k)\}$ $[][]\{Cb^1_s \mid Cb^1_n(S \parallel N)\}$

Here, $\eta = \vee_{i\in I}\eta_i$, $\xi = \vee_{i\in I}\xi_i$ and $Cb^1|Cb^1$ is a conjunction of the two dynamics.

C. Illustrative Example: Continuation 3

Example 4. We continue to explore *Example 1* (Page 3) to demonstrate the usage of guarded choices and algebraic laws. We take the partial parallel processes of *Example 1* without loop for examples. The simplified $Ctrl0$ and $Temp0$ are shown below.

$$Ctrl0 =_{df}$$
$$\left\{\begin{array}{l} @(caron = 1);\ \dot{t} = 1 \textbf{ until } t \geqslant DT + 1;\ \theta_m := \theta; \\ \textbf{if}(\theta_m \geqslant T_{safemax})\textbf{then } coolon := 1;\ \textbf{else } coolon := coolon; \\ \textbf{if}(\theta_m \leqslant T_{safemin})\textbf{then } coolon := 0;\ \textbf{else } coolon := coolon; \\ DT := t; \end{array}\right\}$$

$$Temp0 =_{df}$$
$$\left\{\begin{array}{l} \textbf{if}(coolon == 0) \\ \textbf{then } \dot{\theta} = 1 \textbf{ until } (\theta > T_{MAX} \vee coolon = 1 \vee caron = 0); \\ \textbf{else } \dot{\theta} = -2 \textbf{ until } (\theta < T_{MIN} \vee coolon = 0 \vee caron = 0); \end{array}\right\}$$

For brevity, some program statements are simplified to the following forms. Then, we translate the parallel process $Ctrl0||Temp0$ into the guarded choice form with the aid of algebraic laws given above. We assume that $caron = 1, coolon = 0, DT = 0, t = 0, \theta = 10$ at present.

$$P_1 =_{df} \dot{t} = 1 \textbf{ until } t \geqslant DT + 1; ...; DT := t;$$
$$P_2 =_{df} \theta_m := \theta; ...; DT := t;$$
$$P_3 =_{df} \textbf{if}(\theta_m \geqslant T_{safemax})...; DT := t;$$
$$P_4 =_{df} coolon := coolon; \textbf{if}(\theta_m \leqslant T_{safemin})...; DT := t;$$
$$P_5 =_{df} \textbf{if}(\theta_m \leqslant T_{safemin})...; DT := t;$$
$$P_6 =_{df} coolon := coolon; DT := t;$$
$$P_7 =_{df} DT := t;$$
$$R_1 =_{df} \dot{\theta} = 1 \textbf{ until } (\theta > T_{MAX} \vee coolon = 1 \vee caron = 0);$$
$$R_2 =_{df} \dot{\theta} = -2 \textbf{ until } (\theta < T_{MIN} \vee coolon = 0 \vee caron = 0);$$
$$R_1' =_{df} R_1 \wedge t \in [1, \infty) \qquad R_1'' =_{df} R_1 \wedge t \in [2, \infty)$$

- $Ctrl0||Temp0$: In this parallel composition, the first action of the two parallel components is discrete. From **(guard-1)** and **(cond-1)**, $Ctrl0$ and $Temp0$ are converted into the guarded choice form respectively. Then, we get $Ctrl0||Temp0$ from algebraic law for parallel composition between discrete behaviors in [14].

$$Ctrl0 = [] \{ @(caron = 1) P_1 \}$$

$$Temp0 = [] \left\{ \begin{array}{l} coolon = 0 \& @(x := x) R_1, \\ \neg(coolon = 0) \& @(x := x) R_2 \end{array} \right\}$$

$$= [] \{ true \& @(x := x) R_1 \}$$

$$Ctrl0 || Temp0 = [] \left\{ \begin{array}{l} @(caron = 1)\ (P_1 || Temp0), \\ true \& @(x := x)\ (Ctrl0 || R_1) \end{array} \right\}$$

- $P_1 || Temp0$ and $Ctrl0 || R_1$: In the two parallel composition, the first action of one parallel component is discrete and the other one is continuous. From **(cb-1)**, we transfer P_1 and R_1. From **(par-1-3)** and **(par-2-3)**, we get the two parallel composition below.

$$P_1 = [] \{ CbT^1 P_2 \}$$
$$P_1 || Temp0 = [] \{ true \& @(x := x)\ (P_1 || R_1) \}$$
$$R_1 = [] \{ CbR_1^1 R_1' \}$$
$$Ctrl0 || R_1 = [] \{ @(caron = 1)\ (P_1 || R_1) \}$$

Here,

$$CbT^1 =_{df} (\dot{t} = 1 \textbf{ until } t \geqslant DT + 1) \wedge \delta(t) < 1$$
$$CbR_1^1 =_{df} R_1 \wedge \delta(t) < 1$$

- $P_1 || R_1$: In this parallel composition, the first action of the two parallel components is continuous. From **(par-3-3)**, $P_1 || R_1$ is transformed. As mentioned before, $CbT^1 | CbR_1^1$ is a conjunction of the two dynamics, i.e., the evolution of t and θ follow their own differetial relation without interference respectively.

$$P_1 || R_1 = [] \{ CbT^1 | CbR_1^1\ (P_2 || R_1') \}$$

- $The\ Rest$: Similarly, we continue to expand the parallel composition as below.

$$P_2 = [] \{ true \& @(\theta_m := \theta) P_3 \},\ R_1' = \{ CbR_1^1 R_1'' \}$$
$$P_2 || R_1' = [] \{ true \& @(\theta_m := \theta)\ (P_3 || R_1') \}$$
$$P_3 = [] \{ true \& @(x := x) P_4 \}$$
$$P_3 || R_1' = [] \{ true \& @(x := x)\ (P_4 || R_1') \}$$
$$P_4 = [] \{ true \& @(coolon := coolon) P_5 \}$$
$$P_4 || R_1' = [] \{ true \& @(coolon := coolon)\ (P_5 || R_1') \}$$
$$P_5 = [] \{ true \& @(x := x) P_6 \}$$
$$P_5 || R_1' = [] \{ true \& @(x := x)\ (P_6 || R_1') \}$$
$$P_6 = [] \{ true \& @(coolon := coolon) P_7 \}$$
$$P_6 || R_1' = [] \{ true \& @(coolon := coolon)\ (P_7 || R_1') \}$$

Finally, we need to tranform $P_7 || R_1'$. With **(assign-1)**, we get the guarded choice form of P_7. Since ε denotes an empty program, $\varepsilon || R_1'$ equals to R_1'. We gain the last step of transformation as below.

$$P_7 = [] \{ true \& @(DT := t) \varepsilon \}$$
$$P_7 || R_1' = [] \{ true \& @(DT := t)(\varepsilon || R_1') \}$$
$$= [] \{ true \& @(DT := t) R_1' \}$$

After the above steps, we can finally gain the guard choice form of $Ctrl0 || Temp0$. It implies that programs of our language can be converted into a guard choice form. Further, it indicates that a parallel program can be sequentialized through our algebraic laws.

V. Conclusion and Future Work

In this paper, we elaborated a modeling language for cyber-physical systems based on our previous work [9]. The parallel mechanism between the physical and the cyber of this language is based on shared variables. We also introduced three types of guarded components and five types of guarded choices to support the algebraic parallel expansion laws.

Further, we explored the denotational semantics and algebraic semantics of our language based on the UTP approach [10]. Our main contributions are giving the denotational semantics of the continuous behavior and its composition with the discrete behavior, and summarizing algebraic laws for the parallel composition in which continuous behavior is involved. On this basis, every program of our language can be transformed into a unified form (i.e., guarded choice form), and we can realize the sequentialization of parallel programs with discrete and continuous behaviors. To better understand the usage of our language and its semantics, we also presented an example of a battery management system throughout the paper.

For the future, we plan to study the proof system based on Hoare Logic [15] for our language. We also plan to dive into the semantics linking theory [10] of our language.

References

[1] Lei Bu, Jiawan Wang, Yuming Wu, Xuandong Li: From Bounded Reachability Analysis of Linear Hybrid Automata to Verification of Industrial CPS and IoT. SETSS 2019: 10-43

[2] Yu Wang, Mojtaba Zarei, Borzoo Bonakdarpour, Miroslav Pajic: Statistical Verification of Hyperproperties for Cyber-Physical Systems. ACM Trans. Embed. Comput. Syst. 18(5s): 92:1-92:23 (2019)

[3] Richard Banach, Michael J. Butler, Shengchao Qin, Nitika Verma, Huibiao Zhu: Core Hybrid Event-B I: Single Hybrid Event-B machines. Sci. Comput. Program. 105: 92-123 (2015)

[4] Jifeng He: 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)

[5] Chaochen Zhou, Ji Wang, Anders P. Ravn: A Formal Description of Hybrid Systems. Hybrid Systems 1995: 511-530

[6] Jiang Liu, Jidong Lv, Zhao Quan, Naijun Zhan, Hengjun Zhao, Chaochen Zhou, Liang Zou: A Calculus for Hybrid CSP. APLAS 2010: 1-15

[7] Mauno Rönkkö, Anders P. Ravn, Kaisa Sere: Hybrid action systems. Theor. Comput. Sci. 290(1): 937-973 (2003)

[8] Jifeng He, Qin Li: A Hybrid Relational Modelling Language. Concurrency, Security, and Puzzles 2017: 124-143

[9] Banach, R, Zhu, H. Language evolution and healthiness for critical cyber-physical systems. J Softw Evol Proc. 2021; 33:e2301.

[10] C. A. R. Hoare, He Jifeng: Unifying Theories of Programming. Prentice Hall International Series in Computer Science, 1998

[11] C. A. R. Hoare, Ian J. Hayes, Jifeng He, Carroll Morgan, A. W. Roscoe, Jeff W. Sanders, Ib Holm Sørensen, J. Michael Spivey, Bernard Sufrin: Laws of Programming. Commun. ACM 30(8): 672-686 (1987)

[12] G. Plotkin, A structural approach to operational semantics, Tech. Rep. 19, University of Aahus, 1981 (also published in The Journal of Logic and Algebraic Programming, vols. 60–61, 2004, pp. 17–139)

[13] J. Stoy, Denotational Semantics: The Scott–Strachey Approach to Programming Language, MIT Press, 1977

[14] Huibiao Zhu: Linking the semantics of a multithreaded discrete event simulation language. London South Bank University, UK, 2005

[15] Apt, K.R., de Boer, F.S., Olderog, E.: Verification of Sequential and Concurrent Programs. Texts in Computer Science, Springer (2009)