A Calculus for Hybrid CSP *

Liu Jiang[1], Lv Jidong[2], Quan Zhao[1], Zhan Naijun[1], Zhao Hengjun[1], Zhou Chaochen[1], and Zou Liang[1]

[1] State Key Lab. of Computer Science, Institute of Software, CAS
[2] State Key Lab. of Rail Traffic Control and Safety, Beijing Jiaotong University

Abstract. Hybrid Communicating Sequential Processes (HCSP) is an extension of CSP allowing continuous dynamics. We are interested in applying HCSP to model and verify hybrid systems. This paper is to present a calculus for a subset of HCSP as a part of our efforts in modelling and verifying hybrid systems. The calculus consists of two parts. To deal with continuous dynamics, the calculus adopts differential invariants. A brief introduction to a complete algorithm for generating polynomial differential invariants is presented, which applies DISCOVERER, a symbolic computation tool for semi-algebraic systems. The other part of the calculus is a logic to reason about HCSP process, which involves communication, parallelism, real-time as well as continuous dynamics. This logic is named as Hybrid Hoare Logic. Its assertions consist of traditional pre- and post-conditions, and also Duration Calculus formulas to record execution history of HCSP process.

keywords: Chinese Train Control System, Differential Invariant, DISCOVERER, Duration Calculus, Hoare Logic, Hybrid CSP, Hybrid Logic

1 Introduction

We are interested in modelling and verifying hybrid systems, and take the Level 3 of Chinese Train Control System (CTCS-3) [17] as a case study, which is an informal specification of Chinese high speed train control system that ensures safety and high throughput of trains. There are many reasons to guarantee the high throughput. But our case study only focuses on the analysis and verification of the safety of CTCS-3.

In CTCS-3, there are specifications of 14 scenarios. For example, one of the 14 scenarios specifies that trains are only allowed to move within their current *movement authorities* (MAs) which are determined and updated by *Radio Block Center* (RBC). Hence, the train controller should restrict the movement of the train to ensure that it always runs within its MA with a speed in the scope predefined by the MA. In this scenario, there are continuous dynamics of trains that are described by differential equations, communications between train and RBC, real-time aspects of the movement, etc.

* This work is supported in part by the projects NSFC-60721061, NSFC-90718041, NSFC-60736017, NSFC-60970031, NSFC-60634010 and RCS2008K001.

In order to verify the safety of scenarios, we have to first give a formal model of the scenarios. CSP is a good candidate for modelling communication and parallelism among trains and RBCs. However CSP lacks of mechanisms to describe continuous dynamics of train. In [4, 16], a Hybrid CSP (HCSP) is proposed to model hybrid systems. HCSP introduces into CSP continuous variables, differential equations, and interruptions by events including timeout, communicating, boundary reaching etc. Our experience in using HCSP to model CTCS-3 is quite satisfactory, and the details will be reported in another paper.

This paper is to present a calculus to verify the safety of HCSP process. The calculus consists of two parts. One is to reason about differential equations. We adopt *differential invariants* [10, 12, 11, 3]. In their papers, the authors respectively demonstrate different sufficient conditions to generate/check a differential invariant with respect to a given differential equation. These conditions are useful, but too restrictive to generate some of invariants for our verification of CTCS-3. In [5], we develop an algorithm, which is *complete* in the sense that, if the differential equation is given in polynomials and it has a polynomial inequality (equality) as its invariant, then this algorithm can guarantee the generation of this polynomial invariant. The generation of polynomial differential invariant by this algorithm is supported by a symbolic computation tool for semi-algebraic systems, DISCOVERER [13–15]. This paper gives a brief introduction to this algorithm. Details of the algorithm can be referred to [5].

Another part of the calculus to verify an HCSP process is a logic to deal with communications, parallelism, differential equations, interruptions, timing, etc. In the literature, the Differential Algebraic Dynamic Logic [8] can deal with differential equations through differential invariants. However it does not take into account communication, parallelism, interruption, etc. In this paper we propose a logic which can handle all these issues. Its sequential part is similar to Hoare Logic. For parallel part, since HCSP (like CSP) does not allow memory sharing, we follow the interleaving model for concurrency except communicating. Therefore comparison between sequential processes of a parallel system does not make sense unless synchronization (i.e. communication) happens. Hence, we separate pre- and post-condition for each sequential subsystem of a parallel system, although literally mixing them up is not difficult. A similar idea can be found in [1]. When communication happens, the logic must consider the timing issue of two involved parties. So, in addition to pre- and post-condition, we introduce into Hoare Logic a history formula, which is a Duration Calculus formula[1]. It can treat timing issue and record changes of variable values. The history formula can also help in dealing with interruptions. By interruption we mean a sudden stopping of a process followed by a transition to another one. Reasoning about interruption is really difficult. The paper demonstrates our first attempt to tackle this problem. This logic is based on Hoare Logic, Duration Calculus and Differential Invariants. Thus, we call it Hybrid Hoare Logic.

[1] In [7], Duration Calculus is also used to prove safety critical property for European railways.

2 Hybrid CSP

Hybrid CSP is a modelling language for hybrid systems [4, 16]. HCSP is an extension of CSP, which introduces into CSP differential equations, time constructs, interruptions, etc. It can be used for describing continuous, communicating and real-time behaviour of hybrid systems.

The vocabulary of HCSP includes:

- **Var** is a countable set of discrete variables.
- **Continuous** is a countable set of continuous variables, which are interpreted as continuous functions from time (non-negative reals) to reals. We use **VC** to stand for **Var** $\cup$ **Continuous**.
- **Chan** is a countable set of channels. We use $ch_1, ch_2, \ldots$ to range over channels, and $ch?$ to stand for input, while $ch!$ for output.

Thus, a process of HCSP is defined according to the following grammar:[2]

$$\begin{aligned} P ::= &\ \mathbf{stop} \mid \mathbf{skip} \mid v := e \mid ch?x \mid ch!e \mid \langle F(\dot{s}, s) = 0 \land B\rangle \mid \\ &\ P; Q \mid B \rightarrow P \mid P \trianglerighteq_d Q \mid P \trianglerighteq [\!]_{i\in I}(io_i \rightarrow Q_i) \mid P^* \\ S ::= &\ P \mid P \parallel S \end{aligned}$$

where B is a first order formula over **VC**, and $d > 0$. Intuitively, the above constructs can be understood as follows:

- **stop** does nothing but keeps idle for ever.
- **skip** terminates immediately and does nothing.
- $v := e$ is to assign the value of the expression e to v and then terminates.
- $ch?x$ receives a value to x through the channel ch.
- $ch!e$ sends the value of e to the channel ch, and e is an arithmetic expression of **VC**.
- $\langle F(\dot{s}, s) = 0 \land B\rangle$[3] is a continuous statement. It defines an evolution by a differential equation over s. In fact, s could be a vector of continuous variables, and F be a group of differential equations. B is a first order formula of s, which defines a domain of s in the sense that, if the evolution of s is beyond B, the statement terminates. Otherwise it goes forward.[4]
- $P; Q$ behaves like P first and then behaves like Q after P terminates.
- $B \rightarrow P$ behaves like P if B is true. Otherwise it terminates.
- $P \trianglerighteq_d Q$ behaves like P if P can terminate within d time units. Otherwise, after d time units, it will behave like Q. Here we assume that both P and Q do not contain communications. A **wait** statement, which postpones process behaviour for d time units, can be defined as

$$\mathbf{wait}\ d \mathrel{\widehat{=}} \mathbf{stop} \trianglerighteq_d \mathbf{skip}$$

[2] This is only a subset of HCSP in [4, 16].

[3] This notation is from [9], but here it is interpreted a little differently.

[4] This is written as $\langle F(\dot{s}, s) = 0\rangle \rightarrow \neg B$ in [4, 16].

– $P \trianglerighteq \|_{i\in I}(io_i \rightarrow Q_i)$ behaves like P until a communication in the following context appears. Then it behaves like Q_i immediately after communication io_i occurs. Here I is a non-empty finite set of indices, and $\{io_i \mid i \in I\}$ are input and output statements. We also assume that P does not contain any communications. Furthermore, the *external choice* of CSP can be defined as

$$\|_{i\in I}(io_i \rightarrow Q_i) \mathrel{\widehat{=}} \mathbf{stop} \trianglerighteq \|_{i\in I}(io_i \rightarrow Q_i).$$

– P^* means the execution of P can be repeated arbitrarily finitely many times.
– $P \parallel Q$ behaves as if P and Q are executed independently except that all communications along the common channels between P and Q are to be synchronized. In order to guarantee P and Q having no shared continuous nor discrete variables, and neither shared input nor output channels, we give the following syntactical constraints:

$$\begin{aligned}(\mathbf{VC}(P) \cap \mathbf{VC}(Q)) &= \emptyset\\ (\mathbf{InChan}(P) \cap \mathbf{InChan}(Q)) &= \emptyset\\ (\mathbf{OutChan}(P) \cap \mathbf{OutChan}(Q)) &= \emptyset,\end{aligned}$$

where $\mathbf{VC}(P)$ stands for the set of discrete and continuous variables that indeed appear in P, $\mathbf{InChan}(P)$ ($\mathbf{OutChan}(P)$ for input (output) channels of P.

Examples:

1. Plant Controller: A plant is sensed by a computer periodically (say every d time units), and receives a control (u) from the computer soon after the sensing.

$$((\langle F(u, s, \dot{s}) = 0\rangle \trianglerighteq (c_{p2c}!s \rightarrow \mathbf{skip})); c_{c2p}?u)^* \parallel (\mathbf{wait}\ d; c_{p2c}?x; c_{c2p}!e(x))^*$$

where $\langle F(u, s, \dot{s}) = 0\rangle$ (i.e. $\langle F(u, s, \dot{s}) = 0\rangle \wedge \mathit{true}$ describes the behaviour of the plant. We refer this HCSP process as *PLC* in the rest of the paper.

2. Emergency Brake: A train is moving at an acceleration a until the train reaches an Emergency Brake Intervention speed. Then, it will take an emergency deceleration ($a = -lb$) to return to safe velocity (v_s). During its moving, the train always listens to RBC, if it receives from RBC a message of emergency brake, it decelerates with $-lb$ until it stops. This only shows what a piece of HCSP process joining in the models of CTCS-3 scenarios looks like.

$$\begin{aligned}&(\langle(\dot{s} = v, \dot{v} = a) \wedge (v < v_{ebi})\rangle; \langle(\dot{s} = v, \dot{v} = -lb) \wedge (v \geq v_s)\rangle; ...)\\ &\ \trianglerighteq c_{r2t}?x \rightarrow (x = EB \rightarrow \langle(\dot{s} = v, \dot{v} = -lb) \wedge (v > 0)\rangle); ...\\ &\quad \parallel \mathbf{wait}\ d; (c_{r2t}!EB \rightarrow ... \| ...)\end{aligned}$$

3 Differential Invariants

Verification of HCSP process consists of two parts: an algorithm to generate or check differential invariants and a logic to reason about assertions of the process.

A differential invariant of a differential equation

$$\langle F(s, \dot{s}) = 0 \wedge B \rangle$$

for given initial values of s is a first order formula of s, which is satisfied by the initial values and also by all the values within the area defined by B and reachable by the trajectory of s defined by the differential equation.

In [10], Platzer and Clarke proposed a sufficient condition to check a differential invariant. For differential equation and its domain written as

$$\langle (\dot{s}_1 = f_1, ..., \dot{s}_n = f_n) \wedge B \rangle.$$

$e \leq g$ is a differential invariant of the above differential equations with given initial values of $s_1, ..., s_n$, if the initial values satisfy $e \leq g$, and the first order Lie derivative of e is less than g's, i.e.

$$\sum_{i=1}^{n} \frac{\partial e}{\partial s_i} f_i \leq \sum_{i=1}^{n} \frac{\partial g}{\partial s_i} f_i$$

This condition is useful, but quite rough in checking a differential invariant. For example, $v \leq v_{ebi}$ is a differential invariant of

$$\langle (\dot{s} = v, \dot{v} = a) \wedge v < v_{ebi} \rangle.$$

But it cannot be proved through this sufficient condition unless $a \leq 0$.

When f_js are polynomials in s_i $(i = 1, ..., n)$, and B is a conjunction of polynomial equations and inequalities, the above differential equation is called *semi-algebraic* differential equation. In fact, suppose $\mathbf{s}(t)$ is the trajectory of the above semi-algebraic differential equation starting from a point on the boundary of $e \leq g$, i.e. $e = g$, then the first non-zero higher order Lie derivative of $e(\mathbf{s}(t)) - g(\mathbf{s}(t))$ with respect to t at $t = 0$ provides full information about the evolution tendency of $\mathbf{s}(t)$ with respect to $e \leq g$. If it is less than 0, $\mathbf{s}(t)$ will meet $e \leq g$ as t increases, i.e. $e \leq g$ is an invariant; otherwise, $e \leq g$ will be violated.

Using the above observation, in [5], we proposed a sound and complete method on generating polynomial differential invariants for the semi-algebraic differential equations. The basic idea is to suppose a template of differential invariant $p(s_1, \cdots, s_n, u_1, \cdots, u_m) \sim 0$ first, where p is a polynomial in continuous variables $s_1, \cdots, s_n$ and parameters $u_1, \cdots, u_m$, and $\sim \in \{\geq, >, \leq, <, =, \neq\}$; and then repeatedly compute p's Lie derivative of different order and derive constraints on the parameters according to the signs of the computed derivatives. The hardest part of our method is how to guarantee the termination of the above procedure. By applying some fundamental theories in algebraic geometry, we show that the above procedure of computing derivatives will never be endless. Thus, it is proved that the existence of differential invariants of the predefined template is equivalent to the existence of the solutions of the resulted constraints. Furthermore, the solutions of the constraints construct coefficients of the differential invariant.

Using our method to check the differential invariant of the above example, it amounts to check the validity of

$$\forall v. \begin{pmatrix} (v = v_{ebi} \wedge a \leq 0) \Rightarrow a \leq 0 \wedge \\ (v = v_{ebi} \wedge v < v_{ebi}) \Rightarrow a \leq 0) \end{pmatrix},$$

which is obvious.

In order to generate and solve constraints on the parameters of a template of a differential invariant, we can apply DISCOVERER [13–15], a tool for symbolic computation of semi-algebraic systems, as well as for quantifier elimination [2].

Compared with the existing work on this topic [10, 12, 11, 3], our method is the first sound and complete one to generate polynomial differential invariants for semi-algebraic differential equations. Details are referred to [5].

4 Hybrid Hoare Logic

HCSP adopts message passing communications but rejects memory sharing paradigm. Comparison between variables of different sequential processes of a parallel program makes sense only if they are synchronized. We therefore restrict assertions to formulas of **VC** of each sequential process, although it is not difficult to literally mix them up.

HCSP employs sequential composition of statements, and we follow the traditional pre- and post- conditions of Hoare Logic to deal with sequential composition. A pre-condition specifies the **VC** values right before an execution of a statement, while a post-condition specifies the values immediately after the execution of the statement if it terminates. We use first order formulas of **VC** to express pre- and post- conditions.

However HCSP also includes interruptions by reaching a boundary, by timeout or by a communication. Hence, we need a record of the history of process execution, so that we can retreat to the place where the interruption happens. We take a subset of Duration Calculus (DC) formulas [19, 18] to record an execution history of a process. That is a sequence of DC states over intervals linked together by the modality ($\frown$). It must be very tedious to remember all details of a history, and we need abstraction to develop a simple logic. Computer computation and continuous evolution of plant have different time granularity, and we adopt *super dense computation* [6] to assume computer computation consuming zero time. This agrees with the abstraction of DC: a state being present over an interval means that the state holds *almost everywhere* in the interval. This abstraction has many advantages. But in some cases it may damage the connection after an interruption. So, through DC events [18], history can still remember the points where value changes do happen, although it may neglect the particular values at those points.

4.1 Subset of DC Formulas

As indicated before, we will use a subset of DC formulas to record execution history of HCSP process. The formula in this subset is denoted as *HF* (*history* *formula*) and given as follows.

$$\begin{array}{rl} HF ::= & l < T \mid l = T \mid l > T \mid \uparrow_X \mid \lceil S \rceil \\ & \mid HF^\frown HF \mid HF \wedge HF \mid HF \vee HF \end{array}$$

where l stands for interval length, X is a subset of **VC**, S is a first order formula of **VC**, and $T \geq 0$.

$\uparrow_X$ is an event to mean changes of variables in X taking place at a time point. The axioms and rules can be copied from the event calculus in [18]. But in order to maintain this information unaltered during deductions, we only list two of them as axioms. The others can be used as antecedence when needed. These two axioms are

$$\begin{array}{l} \uparrow_\emptyset \Leftrightarrow \ (l = 0) \\ \uparrow_X {}^\frown \uparrow_Y \Leftrightarrow \uparrow_{X \cup Y} \end{array}$$

$\lceil S \rceil$ means S true almost everywhere over an interval. It follows all the theorems of $\lceil S \rceil$ in [18], such as

$$\begin{array}{l} \lceil S \rceil^\frown \lceil S \rceil \ \Leftrightarrow \ \lceil S \rceil, \\ \lceil S \rceil^\frown (l = 0) \ \Leftrightarrow \ \lceil S \rceil, \\ \text{etc.} \end{array}$$

All proofs for HF are given in DC (plus the above two axioms for $\uparrow_X$), and will not be explicitly indicated. For example, we can prove in DC:

$$\begin{array}{l} \mathit{false} \ \Leftrightarrow \ (l < 0) \\ \mathit{true} \ \Leftrightarrow \ (l = 0) \vee (l > 0) \end{array}$$

Since an interruption may occur at any time during process execution, to locate it we define prefix closure of HF and denote it as $HF^<$.

$$\begin{array}{ll} (l < T)^< & =_{df} (l < T) \\ (l = T)^< & =_{df} (l \leq T) \\ (l > T)^< & =_{df} \mathit{true} \\ (\uparrow_X)^< & =_{df} \uparrow_X \\ \lceil S \rceil^< & =_{df} (l = 0) \vee \lceil S \rceil \\ (HF_1^\frown HF_2)^< & =_{df} \begin{cases} \mathit{false} & \text{if } HF_2 \Rightarrow \mathit{false} \\ (HF_1)^< \vee HF_1^\frown (HF_2)^< & \text{otherwise} \end{cases} \\ (HF_1 \wedge HF_2)^< & =_{df} \begin{cases} \mathit{false} & \text{if } HF_1 \wedge HF_2 \Rightarrow \mathit{false} \\ (HF_1)^< \wedge (HF_2)^< & \text{otherwise} \end{cases} \\ (HF_1 \vee HF_2)^< & =_{df} (HF_1)^< \vee (HF_2)^< \end{array}$$

It is obvious that the prefix closure of any formula of the subset still belongs to it. From the above definition, we can prove

$$\begin{array}{l} \mathit{true}^< \Leftrightarrow \mathit{true} \\ \mathit{false}^< \Leftrightarrow \mathit{false} \end{array}$$

4.2 Assertions

An assertion of Hybrid Hoare Logic consists of four parts: precondition, process, postcondition and history, written as

$$\{\mathit{Pre}\}P\{\mathit{Post}; \mathit{HF}\}$$

where Pre specifies values of $\mathbf{VC}(P)$ before an execution of P, Post specifies $\mathbf{VC}(P)$ values when it terminates, and HF is a formula of $\mathbf{VC}(P)$ from the DC subset to describe the execution history of P, which includes differential invariants of P. In Hoare Logic, a loop invariant joins in postcondition of the loop, so does in this Hybrid Hoare Logic. HCSP has three kinds of interruptions: boundary interruption, e.g. $\langle F(\dot{s}, s) = 0 \wedge B\rangle$, timeout interruption, e.g. $P \trianglerighteq_d Q$ and communication interruption, e.g. $P \trianglerighteq \|_{i\in I}(io_i \to Q_i)$. For these three kinds of interruptions, HF has to join in reasoning. In HF, $\uparrow_X$ indicates that the changes of variables in X may take place at this point, and reasoning about assertions at this point should not rely on these variables.

For a parallel process, say $P_1 \parallel ... \parallel P_n$, the assertion becomes

$$\{\mathit{Pre}_1, ..., \mathit{Pre}_n\}P_1 \parallel ... \parallel P_n\{\mathit{Post}_1, ..., \mathit{Post}_n; \mathit{HF}_1, ..., \mathit{HF}_n\}$$

where $\mathit{Pre}_i, \mathit{Post}_i, \mathit{HF}_i$ are (first order or DC) formulas of $\mathbf{VC}(P_i)$ $(i = 1, ..., n)$.

Another role of HF is to specify real-time (continuous) property of an HCSP process, while Pre and Post can only describe its discrete behaviour. HF therefore bridges up the gap between discrete and continuous behaviour of the process. For example, we may want the plant controller example (PLC) in Section 2 stable after T time units, i.e. after T time units the distance between the trajectory of s and its target s_{targ} must be small. This can be specified through the following assertion.

$$\begin{array}{l}\{s = s_0 \wedge u = u_0 \wedge \mathit{Ctrl}(u_0, s_0), \mathit{Pre}_2\}\mathit{PLC}\\ \quad\{\mathit{Post}_1, \mathit{Post}_2; (l = T)^\frown\lceil \mid s - s_{targ} \mid \leq \epsilon\rceil, \mathit{HF}_2\}\end{array}$$

where $\mathit{Ctrl}(u, s)$ may express a controllable property, and the other formulas are not elaborated here.

4.3 Axioms and Rules

We do not list all axioms and rules for all HCSP processes, but explain our idea how to establish this logic. Say, in this subsection we only use a parallel process consisting of two sequential ones to demonstrate the logic.

1. **Monotonicity**

$$\begin{array}{l}\text{If } \{\mathit{Pre}_1, \mathit{Pre}_2\}P_1 \parallel P_2\{\mathit{Post}_1, \mathit{Post}_2; \mathit{HF}_1, \mathit{HF}_2\},\\ \text{and } \mathit{Pre}'_i \Rightarrow \mathit{Pre}_i, \mathit{Post}_i \Rightarrow \mathit{Post}'_i, \mathit{HF}_i \Rightarrow \mathit{HF}'_i (i = 1, 2),\\ \text{then } \{\mathit{Pre}'_1, \mathit{Pre}'_2\}P_1 \parallel P_2\{\mathit{Post}'_1, \mathit{Post}'_2; \mathit{HF}'_1, \mathit{HF}'_2\}\end{array}$$

where we use first order logic to reason $\mathit{Pre}'_i \Rightarrow \mathit{Pre}_i$ and $\mathit{Post}_i \Rightarrow \mathit{Post}'_i$, but use DC (plus the two axioms for $\uparrow_X$) to reason $\mathit{HF}_i \Rightarrow \mathit{HF}'_i$. From now on we will not repeatedly mention this.

2. **Case Analysis**

   If $\{Pre_{1i}, Pre_2\}P_1 \parallel P_2\{Post_1, Post_2; HF_1, HF_2\}$ $(i = 1, 2)$,
   then $\{Pre_{11} \vee Pre_{12}, Pre_2\}P_1 \parallel P_2\{Post_1, Post_2; HF_1, HF_2\}$

   Symmetrically,

   If $\{Pre_1, Pre_{2i}\}P_1 \parallel P_2\{Post_1, Post_2; HF_1, HF_2\}$ $(i = 1, 2)$,
   then $\{Pre_1, Pre_{21} \vee Pre_{22}\}P_1 \parallel P_2\{Post_1, Post_2; HF_1, HF_2\}$

3. **Parallel vs Sequential**
   These two rules show a simple relation between assertions of a parallel process and its sequential components that can ease a proof.

   If $\{Pre_1, Pre_2\}P_1 \parallel P_2\{Post_1, Post_2; HF_1, HF_2\}$
   then $\{Pre_i\}P_i\{Post_i; HF_i\}$ $(i = 1, 2)$

   and

   If $\{Pre_i\}P_i\{Post_i; HF_i\}$ $(i = 1, 2)$,
   and P_i $(i = 1, 2)$ do not contain communication,
   then $\{Pre_1, Pre_2\}P_1 \parallel P_2\{Post_1, Post_2; HF_1, HF_2\}$

4. **Stop**
   stop does nothing, and never terminates. So, **stop** will keep any precondition true for ever. Hence, for any $r \geq 0$,

   $$\{Pre\}\mathbf{stop}\{Pre; \lceil Pre \rceil \wedge (l > r)\}$$

5. **Skip**

   $$\{Pre\}\mathbf{skip}\{Pre; l = 0\},$$

   where by $l = 0$ we assume that, in comparison with physical device, computation takes no time (i.e. *supper dense computation* [6])

6. **Assignment**

   $$\{Pre[e/x]\}x := e\{Pre, \uparrow_x\}$$

   The precondition and postcondition are copied from Hoare Logic. Here we use $\uparrow_x$ as its history to indicate that, a change of x takes place at this time point, although the history does not record the values of x before and after the change.

7. **Communication**
   Since HCSP rejects variable sharing, a communication looks like the output party $(P_1; ch!e)$ assigning to variable x of the input one $(P_2; ch?x)$ a value (e). Besides, in order to synchronize both parties, one may have to wait for another. During the waiting of P_i, $Post_i$ must stay true $(i = 1$ or $2)$.

Furthermore, when we conclude range of the waiting time, we need to reduce $\uparrow_X$ to $(l = 0)$.

$$\begin{array}{l}
\text{If } \{Pre_1, Pre_2\}P_1 \parallel P_2\{Post_1, Post_2; HF_1, HF_2\}, \\
\quad Post_1 \Rightarrow G(e), \\
\text{and } \Box(\bigwedge_{X\subseteq \mathbf{VC}} \uparrow_X \Rightarrow (l=0)) \ \wedge \ (((HF_1^\frown(\lceil Post_1\rceil)^<) \wedge HF_2) \vee \\
\qquad (HF_1 \wedge (HF_2^\frown(\lceil Post_2\rceil)^<))) \ \Rightarrow \ Rg(l) \\
\quad\ \text{where } Rg(l) \text{ is an arithmetic formula of } l \text{ to define its range} \\
\text{then } \{Pre_1, Pre_2\}(P_1; ch!e) \parallel (P_2; ch?x) \\
\qquad \{Post_1, G(x) \wedge \exists x Post_2; HF_1^\frown(\lceil Post_1\rceil)^< \wedge Rg(l), \\
\qquad\ (HF_2^\frown(\lceil Post_2\rceil)^< \wedge Rg(l))^\frown \uparrow_x\}
\end{array}$$

Example:
If

$$\begin{array}{l}
\{Pre_1, Pre_2\}P_1 \parallel P_2 \\
\quad \{y = 3, x = 1; (\lceil y = 0\rceil \wedge (l = 3))^\frown \uparrow_y, \lceil x = 0\rceil \wedge (l = 5)^\frown \uparrow_x\},
\end{array}$$

we want to deduce through this rule

$$\{Pre_1, Pre_2\}P_1; ch!y \parallel P_2; ch?x\{Post_3, Post_4; HF_3, HF_4\}.$$

Since $(y = 3) \Rightarrow (3 = 3)$ and

$$\begin{array}{l}
\Box \bigwedge_{X\subseteq\{x,y\}}(\uparrow_X \Rightarrow (l = 0)) \wedge \\
\ ((\lceil y = 0\rceil \wedge (l = 3))^\frown \uparrow_y^\frown ((l = 0) \vee \lceil y = 3\rceil)) \wedge ((\lceil x = 0\rceil \wedge (l = 5))^\frown \uparrow_x) \\
\quad \Rightarrow (l = 5),
\end{array}$$

we can conclude that $Post_3$ is $y = 3$, $Post_4$ is $x = 3$,
HF_3 is

$$((\lceil y = 0\rceil \wedge (l = 3))^\frown \uparrow_y^\frown \lceil y = 3\rceil) \wedge (l = 5),$$

and HF_4 is

$$(l = 5)^\frown \uparrow_x^\frown \uparrow_x$$

which is equivalent to

$$(l = 5)^\frown \uparrow_x$$

by the axioms of $\uparrow_X$.

8. **Continuous**
This is about $\langle F(\dot{s}, s) = 0 \wedge B\rangle$, where s can be a vector and F be a group of differential equations, such as

$$\langle(\dot{s}_1 = f_1, ..., \dot{s}_n = f_n) \wedge B\rangle.$$

As indicated in Section 3, in this paper we only deal with *semi-algebraic* differential equations and polynomial differential invariants. That is, f_js are polynomials in s_i $(i = 1, ..., n)$, B is a conjunction of polynomial equations

and inequalities of s_i $(i = 1, ..., n)$, and differential invariants are also restricted to polynomial equations and inequalities.
We have two rules for semi-algebraic differential equations. The first one is about differential invariant. Given a polynomial differential invariant *Inv* of $\langle F(\dot{s}, s) = 0 \wedge B\rangle$ with initial values satisfying *Init*

$$\begin{array}{l} \text{If } \mathit{Init} \Rightarrow \mathit{Inv}, \\ \text{then } \{\mathit{Init} \wedge \mathit{Pre}\}\langle F(\dot{s}, s) = 0 \wedge B\rangle\{\mathit{Pre} \wedge \mathbf{Close}(\mathit{Inv}) \wedge \mathbf{Close}(\neg B); \\ \quad (l = 0) \vee \lceil \mathit{Inv} \wedge \mathit{Pre} \wedge B\rceil\} \end{array}$$

where *Pre* does not contain s, $\mathbf{Close}(G)$ stands for the *closure* of G, [5] and $(l = 0)$ in the history is to record the behaviour when the initial values satisfy $\neg B$ at very beginning.
The second rule is about explicit time.

$$\begin{array}{l} \text{If } \{\mathit{Pre}\}\langle F(\dot{s}, s) = 0 \wedge B\rangle\{\mathit{Post}; \mathit{HF}\} \\ \text{and } \{\mathit{Pre} \wedge t = 0\}\langle(F(\dot{s}, s) = 0, \dot{t} = 1) \wedge B\rangle\{\mathit{Rg}(t); \mathit{HF}'\}, \\ \text{then } \{\mathit{Pre}\}\langle F(\dot{s}, s) = 0 \wedge B\rangle\{\mathit{Post}; \mathit{HF} \wedge \mathit{Rg}(l)\} \end{array}$$

where t is a clock to count the time, and $\mathit{Rg}(t)$ is an arithmetic formula as explained in the rule for communication.

Example:
We know from Section 3 that $v \leq v_{ebi}$ is an invariant of

$$\langle(\dot{s} = v, \dot{v} = a) \wedge v < v_{ebi}\rangle.$$

Thus, by the first rule

$$\begin{array}{l} \{(v = v_0 \leq v_{ebi})\}\langle(\dot{s} = v, \dot{v} = a) \wedge v < v_{ebi}\rangle \\ \quad \{(v \leq v_{ebi}) \wedge (v \geq v_{ebi}); (l = 0) \vee \lceil(v \leq v_{ebi}) \wedge (v < v_{ebi})\rceil\} \end{array}$$

In addition, we can prove that, if the initial values are $v = v_0$ and $t = 0$, and we assume $p \geq a \geq w$, then

$$((v_0 + wt) \leq v \leq (v_0 + pt)) \wedge (v \leq v_{ebi})$$

is an invariant of $\langle(\dot{s} = v, \dot{v} = a, \dot{t} = 1) \wedge v < v_{ebi}\rangle$. So under the assumption $(p \geq a \geq w)$

$$\begin{array}{l} \{(v = v_0 \leq v_{ebi}) \wedge (t = 0)\}\langle(\dot{s} = v, \dot{v} = a, \dot{t} = 1) \wedge v < v_{ebi}\rangle \\ \quad \{(v = v_{ebi}) \wedge ((v_0 + wt) \leq v \leq (v_0 + pt)); \\ \qquad (l = 0) \vee \lceil(v < v_{ebi}) \wedge ((v_0 + wt) \leq v \leq (v_0 + pt))\rceil\} \\ \{(v = v_0 \leq v_{ebi}) \wedge (t = 0)\}\langle(\dot{s} = v, \dot{v} = a, \dot{t} = 1) \wedge v < v_{ebi}\rangle \\ \quad \{\frac{v_{ebi} - v_0}{w} \geq t \geq \frac{v_{ebi} - v_0}{p}; \mathit{true}\} \end{array}$$

Therefore assuming $(p \geq a \geq w)$ we can have

$$\begin{array}{l} \{(v = v_0 \leq v_{ebi})\}\langle(\dot{s} = v, \dot{v} = a) \wedge v < v_{ebi}\rangle \\ \quad \{(v = v_{ebi}); \lceil(v < v_{ebi})\rceil \wedge (\frac{v_{ebi} - v_0}{w} \geq l \geq \frac{v_{ebi} - v_0}{p})\} \end{array}$$

[5] When G is constructed by polynomial inequalities through $\wedge$ and $\vee$, $\mathbf{Close}(G)$ can be obtained from G by replacing $<$ (and $>$) with $\leq$ (and $\geq$) in G.

9. **Sequential**

   If $\{Pre_1, Pre_2\}P_1 \parallel P_2\{Post_1, Post_2; HF_1, HF_2\}$,
   $\{Post_i\}P_{i+2}\{Post_{i+2}; HF_{i+2}\} \quad (i = 1, 2)$,
   and both P_3 and P_4 do not contain communication,
   then $\{Pre_1, Pre_2\}P_1; P_3 \parallel P_2; P_4\{Post_3, Post_4; HF_1^\frown HF_3, HF_2^\frown HF_4\}$.

10. **Timeout**
    We have two rules for $P_1 \unrhd_d P_2$. One is for the case when P_1 terminates before d time units. Another is for the timeout. The first one is

    If $\{Pre\}P_1\{Post; HF\}$
    and $(\Box(\bigwedge_{X\subseteq \mathbf{VC}} \uparrow_X \Rightarrow (l = 0)) \wedge HF) \Rightarrow (l < d)$
    then $\{Pre\}P_1 \unrhd_d P_2\{Post; HF\}$

    The second one is more complicated. The execution of P_1 is interrupted after d time units, and then P_2 starts its execution. Therefore the postcondition of P_1 cannot be used for this transition, and we have to use its history at time d.

    If $\{Pre_1\}P_1\{Post_1; HF_1\}$,
    $\{Pre_2\}P_2\{Post_2; HF_2\}$,
    $(\Box(\wedge_{X\subseteq \mathbf{VC}} \uparrow_X \Rightarrow (l = 0)) \wedge HF_1) \Rightarrow (l \geq d)$
    and $G \;\Rightarrow\; Pre_2$
    then $\{Pre_1\}P_1 \unrhd_d P_2\{Post_2; HF^{*\frown} HF_2\}$

    where G and HF^* are constructed as follows. Choose an HF^* in the form of

    $$\bigvee_{i=1}^{n} HF_i^{*\frown}(\lceil G_i \wedge F_i \rceil \wedge Rg_i(l))^\frown \uparrow_{X_i}$$

    according to the following two criteria. If no variable of G_i is included in X_i $(i = 1, ..., n)$, then we let G be $\bigvee_{i=1}^n G_i$.
    The first criterion to choose HF^* is to guarantee that HF^* does not lose any $\uparrow_X$ in HF_1. That is, we have to prove

    $$HF^* \;\Rightarrow\; HF_1^{<}, \text{ and}$$
    $$\Box(\bigwedge_{((X\neq Y)\wedge X,Y\subseteq \mathbf{VC})} \neg(\uparrow_X \wedge \uparrow_Y))$$
    $$\quad \Rightarrow\; \bigwedge_{i=1}^{n} \neg((HF_i^{*\frown}(\lceil G_i \wedge F_i \rceil \wedge Rg_i(l))^\frown \uparrow_{Y_i}) \wedge HF_1^{<})$$

    for any $Y_i \supset X_i$.
    The second criterion is about the length of HF^* and another direction of the implication between HF_1 and HF^*. That is

    $$\Box(\bigwedge_{X\subseteq \mathbf{VC}} \uparrow_X \Rightarrow (l = 0)) \wedge HF^* \;\Rightarrow\; (l = d), \text{ and}$$
    $$\Box(\bigwedge_{X\subseteq \mathbf{VC}} \uparrow_X \Rightarrow (l = 0)) \wedge HF_1^{<} \wedge (l = d) \;\Rightarrow\; HF^*$$

    In summary, HF^* is a part of $(HF_1^{<} \wedge (l = d))$ that includes all information about variable changes at time points until d (inclusive), and G therefore catches the last states of $(HF_1^{<} \wedge (l = d))$, which do not change at time d.

Examples:

(a) **wait** d $(d > 0)$

$$\{Pre\}\textbf{wait }d\{Pre; \lceil Pre \rceil \wedge (l = d)\}$$

where **wait** d is defined as $\textbf{stop} \trianglerighteq_d \textbf{skip}$. Its proof can be given as follows. Since

$\{Pre\}\textbf{stop}\{Pre; \lceil Pre \rceil \wedge (l > d)\}$,
$\{Pre\}\textbf{skip}\{Pre; (l = 0)\}$,
$(l > d) \;\Rightarrow\; (l \geq d)$,
and we can choose HF^* as $(\lceil Pre \rceil \wedge (l = d))$, and hence, G as Pre,

we can conclude

$$\{Pre\}\textbf{stop} \trianglerighteq_d \textbf{skip}\{Pre; (\lceil Pre \rceil \wedge (l = d))^\frown(l = 0)\}.$$

That is

$$\{Pre\}\textbf{wait }d\{Pre; \lceil Pre \rceil \wedge (l = d)\}.$$

(b) Let P be

$$z := 0; \textbf{wait } 3; y := 3; \textbf{wait } 2$$

and we can prove

$$\begin{array}{l}\{y = 1, z = 2\}P\{(z = 0) \wedge (y = 3);\\ \quad \uparrow_z^\frown (\lceil (y = 1) \wedge (z = 0) \rceil \wedge (l = 3))^\frown \uparrow_y\\ \quad\quad ^\frown(\lceil (z = 0) \wedge (y = 3) \rceil \wedge (l = 2))\}\end{array}$$

and denote the history formula of P as $HF(P)$. For $P \trianglerighteq_3 Q$, P is interrupted after being executed 3 time unit. Let HF^* be

$$\uparrow_z^\frown (\lceil (y = 1) \wedge (z = 0) \rceil \wedge (l = 3))^\frown \uparrow_y$$

We can prove

$HF^* \;\Rightarrow\; HF(P)^{<}$,
$\Box \bigwedge_{(x \neq y)} \neg(\uparrow_{\{x,y\}} \wedge \uparrow_y)$
$\quad \Rightarrow\; \neg((\uparrow_z^\frown (\lceil (y = 1) \wedge (z = 0) \rceil \wedge (l = 3))^\frown \uparrow_{\{x,y\}}) \wedge HF(P)^{<})$,
$\Box(((\uparrow_y \vee \uparrow_z) \Rightarrow (l = 0)) \wedge HF^*) \;\Rightarrow\; (l = 3)$, and
$\Box((\uparrow_y \vee \uparrow_z) \Rightarrow (l = 0)) \wedge HF(P)^{<} \wedge (l = 3) \;\Rightarrow\; HF^*$.

So, G is $(z = 0)$ (and $(y = 1)$ is not involved), and $(z = 0)$ can therefore be used as a precondition of Q.

11. **Choice**
This is about inference rule for $(P \trianglerighteq \|_{i \in I}(io_i \rightarrow Q_i))$. It involves communication interruption which happens randomly, and must be difficult to deal with. If we assume that from one party of the communication we can derive a range of the interruption time, then we can use the history to support the reasoning. Of course we also have to take into account the waiting time of two parties. But all those ideas have been explained before. Thus we omit them here.

12. **Repetition**
We can pick up rules from the literature for the repetition. Here we only show a rule which ends off an assertion reasoning.

If $\{Pre_1, Pre_2\}P_1 \parallel P_2\{Pre_1, Pre_2; HF_1, HF_2\}$,
$((\Box \bigwedge_{X \subseteq \mathbf{VC}}(\uparrow_X \Rightarrow (l = 0))) \wedge HF_i) \;\Rightarrow\; (D_i \wedge (l = T))$ $(i = 1, 2,\; T > 0)$,
and $D_i^\frown D_i \;\Rightarrow\; D_i$,
then $\{Pre_1, Pre_2\}P_1^* \parallel P_2^*\{Pre_1, Pre_2; D_1, D_2\}$

where T is the time consumed by both P_1 and P_2 that can guarantee the synchronisation of the starting point of each repetition.

5 Conclusion

This paper sketches part of our on-going efforts in formally modelling and verifying hybrid systems. We choose a subset of HCSP for modelling, and explain our idea to develop a calculus for this subset, including an improvement of generating and checking differential invariants. So far we are not sure whether this subset is good enough to model interesting hybrid systems, say CTCS-3, and neither the calculus is powerful enough in verifying its safety. Although this is a subset of HCSP, it is quite complicated already in terms of verification. In particular, it includes random interruptions which are hard to handle. Our idea is to use history of execution which records the continuous evolution of process as well as the discrete change of its variables. The calculus tries to leave details as far as we can. Its soundness is not trivial. For this, we need formal semantics of HCSP. A DC-based denotational semantics for HCSP has been established in [16]. Recently, we defined an operational semantics for HCSP, and will check the soundness of the logic against the semantics formally as a future work.

References

1. K. Apt, F. de Boer, and E.-R. Olderog. Verfication of Sequential and Concurrent Programs. Springer, ISBN 978-1-184882-744-8, 2009.
2. G.E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. *Automata Theory and Formal Languages,* LNCS 33, pp. 134–183, 1975.
3. S. Gulwani and A. Tiwari. Constraint-based approach for analysis of hybrid systems. In Proc. of *CAV 2008*, LNCS, 5123, pp. 190–203, 2008.
4. J. He. From CSP to hybrid systems. In the proc. of A Classical Mind: Essays in Honour of C. A. R. Hoare, Prentice-Hall International Series In Computer Science, ISBN:0-13-294844-3, pp. 171-189. 1994.
5. J. Liu, N. Zhan and H. Zhao. A complete method for generating polynomial differential invariants. *Technical Report of State Key Lab. of Comp. Sci., ISCAS-LCS-10-15*, 2010.
6. Z. Manna and A. Pnueli. Models of reactitivity. Acta Informatica, 30(7):609-678. 1993.

7. R.-R. Olderog and H. Dierks. Real-Time Systems: Formal Secification and Automatic Verification. Cambridge University Press, 2008.
8. A. Platzer. Differential-algebraic dynamic logic for differential-algebraic programs. *J. of Logic and Computation*, 20(1): 309-352, 2010.
9. A. Platzer. Differential dynamic logic for hybrid systems. *J. of Automated Reasoning*, 41: 143-189, 2007.
10. A. Platzer and E. M. Clarke. Computing differential invariants of hybrid systems as fixedpoints. In Proc. of *CAV 2008*, LNCS 5123, pp. 176-189, 2008.
11. S. Prajna and A. Jadbabaie. Safety verification of hybrid systems using barrier certificates. In Proc. of *HSCC 2004*, LNCS 2993, pp. 477–492, 2004.
12. S. Sankaranarayanan, H. B. Sipma, and Z. Manna. Constructing invariants for hybrid systems. In Proc. of *HSCC 2004*, LNCS 2993, pp. 539–554, 2004.
13. B. Xia. DISCOVERER: A tool for solving semi-algebraic systems, *Software Demo at ISSAC 2007*, Waterloo, July 30, 2007. Also: *ACM SIGSAM Bulletin*, 41(3):102–103, Sept., 2007.
14. L. Yang. Recent advances on determining the number of real roots of parametric polynomials. *J. Symbolic Computation*, 28:225–242, 1999.
15. L. Yang, X. Hou and Z. Zeng. A complete discrimination system for polynomials. *Science in China (Ser. E)*, 39:628–646, 1996.
16. C. Zhou, J. Wang, A. Ravn. A formal description of hybrid systems. In the proc. of *Hybrid Systems'95*, LNCS 1066, pp. 511-530. 1995
17. S. Zhang. The General Technical Solutions to Chinese Train Control System at Level 3 (CTCS-3). China Railway Publisher, 2008.
18. C. Zhou and M. Hansen. Duration Calculus: A Formal Approach to Real-Time Systems. Springer, ISBN 3-540-40823-1, 2004.
19. C. Zhou, C. A. R. Hoare, and A. Ravn. A calculus of durations. *Information Processing Letters*, 1991,40(5):269–276.