Super-Dense Computation in Verification of Hybrid CSP Processes

Dimitar P. Guelev[2], Shuling Wang[1], Naijun Zhan[1], and Chaochen Zhou[1]

[1] State Key Laboratory of Computer Science, Institute of Software
Chinese Academy of Sciences, Beijing, China
[2] Institute of Mathematics and Informatics, Bulgarian Academy of Sciences
Sofia, Bulgaria

Abstract. Hybrid Communicating Sequential Processes (HCSP) extends CSP to include differential equations and interruptions. We feel comfortable in our experience with HCSP to model scenarios of the Level 3 of Chinese Train Control System (CTCS-3), and to define a formal semantics for Simulink. The Hoare style calculus of [5] proposes a calculus to verify HCSP processes. However it has an error with respect to super-dense computation. This paper is to establish another calculus for a subset of HCSP, which uses Duration Calculus formulas to record program history, negligible time state to denote super-dense computation and semantic continuation to avoid infinite interval. It is compositional and sound.

Keywords: Hybrid system, differential invariant, Hybrid CSP, Duration Calculus, super-dense computation, Hybrid Hoare Logic

1 Introduction

Hybrid system combines discrete control and continuous evolution. A continuously evolving plant with discrete control is a typical example. The behaviour of the plant can be defined by a differential equation, say $F(\dot{s}, s, u) = 0$. A computer samples the state of the plant every d time units through a *sensor*, calculates its control parameter u according to the sensed state s and sends back to the plant through an *actuator*. Communicating Sequential Processes (CSP, [4]) provides channels to model the sensor and the actuator, and parallelism to model interaction between the computer and the plant. However CSP lacks a construct to model physical behaviour of the plant. [3, 14] propose a Hybrid CSP (HCSP) and suggest to use HCSP to model hybrid systems. HCSP introduces into CSP continuous variables, differential equations, and interruptions by boundary, timeout and communication. Our experience in using HCSP to describe the scenarios of Level 3, Chinese Train Control System (CTCS-3) [15] and give a formal semantics of Simulink [16] is quite satisfactory, and will be reported in other papers.

This paper presents a compositional Hoare style calculus to verify properties of HCSP processes. The calculus has to meet two challenges. The first one is

to reason about differential equations. We adopt *differential invariants* from [8, 9]. An algorithm which generates a polynomial (in)equality invariant from a polynomial differential equation is developed in [6]. The algorithm is *complete* as it always produces an invariant, provided that one exists. The generation of the polynomial invariant is supported by a symbolic computation tool for semi-algebraic system, DISCOVERER, which is based on the theory invented in [11, 10].

Another challenge is how to accommodate *super-dense computation* in the calculus. By *super-dense computation* we mean that computer is much faster than other physical devices and computation time of a computer is therefore negligible, although the temporal order of computations is still there. In the plant control example, the control parameter sent from the computer is supposed to control the state sensed before. The computation time for calculating the new control parameter is neglected.

A Hoare style logic for reasoning about HCSP is firstly proposed by [5], which simply uses the *chop* modality of Duration Calculus (DC) [13] to describe the sequential composition, and point intervals to describe super-dense computations. Unfortunately chop degenerates into *conjunction* at a point interval, and the temporal order of computations disappears. Hence, the monotonicity rule cannot be maintained, and it forms an error of [5]. In this paper we use a dedicated DC state variable N to mark *negligible time*. This idea can be traced back to [7, 1]. It is also used in [2], where another Hoare style calculus for HCSP is proposed with different semantic specification. Another approach to deal with super-dense computation is to use pre- and post-conditions as well as history formulas as done in [5], but delete point values in the history formulas in order to maintain the monotonicity rule (see [12] for details).

We also use Hoare triple in our calculus. Triples have the form

$$\{PreH\}Sys\{PostH\},$$

where Sys is an HCSP process, and $PreH$ and $PostH$ are DC formulas to express properties of the *pre-history* and *post-history* of the execution of Sys. A dedicated propositional letter C is used to indicate whether a behaviour defined by a history formula can be further extended.

Structure of the paper In Section 2, we will introduce the language Hybrid CSP briefly, and then present the calculus for reasoning about Hybrid CSP in Section 3 and Section 4. Finally, we conclude the paper by a discussion about the calculus.

2 Hybrid CSP

Hybrid CSP is an extension of CSP with differential equations and interruptions to model behaviour of hybrid systems.

Notation:

HCSP vocabulary includes:

– a countable set of discrete and continuous variables, which are interpreted as functions from time (non-negative reals) to reals, and
– a countable set of channel names.

HCSP is defined according to the following grammar, where v stands for a variable, s and $\dot{s}$ stand for a vector of variables and their time derivatives, ch stands for channel name, I stands for a non-empty finite set of indices, e and B are arithmetical expression and boolean expression of variables, and d is a positive (real) constant.

$$\begin{aligned} P ::= & \text{ skip} \mid v := e \mid \text{wait } d \mid P; Q \mid B \to P \mid P \sqcup Q \\ & \mid ch?x \mid ch!e \mid [\!]_{i \in I}(io_i \to P_i) \\ & \mid \langle F(\dot{s}, s) = 0 \& B \rangle \mid \langle F(\dot{s}, s) = 0 \& B \rangle \trianglerighteq_d P \\ & \mid \langle F(\dot{s}, s) = 0 \& B \rangle \trianglerighteq [\!]_{i \in I}(io_i \to P_i) \\ Sys ::= & P \mid P^* \mid Sys_1 \parallel Sys_2 \end{aligned}$$

Here follows the meaning of each construct:

– skip does nothing and terminates immediately.
– $v := e$ is an atomic assignment.
– wait d does nothing and terminates right after d time units.
– $P; Q$ is the sequential composition of P and Q. It behaves as P first, and then Q after P terminates.
– $B \to P$ behaves like P if B is true. Otherwise it terminates immediately.
– $P \sqcup Q$ is the *internal choice* of CSP. We include this operator to simulate non-deterministic actions.
– $ch?x$ inputs a value over channel ch and stores in x.
– $ch!e$ sends the value of e over channel ch. Here we assume the synchronous communication as defined in CSP.
– $[\!]_{i \in I}(io_i \to P_i)$ is the *external choice* of CSP. An occurrence of io_i can lead to the execution of P_i, where io_i stands for an input or output.
– $\langle F(\dot{s}, s) = 0 \& B \rangle$ defines a bounded evolution of the differential equation F over s. B is a boolean expression of s, which defines a domain of s in the sense that, if the evolution of s as defined by $F(\dot{s}, s) = 0$ is beyond B, the statement terminates. Otherwise it goes forward.
– $\langle F(\dot{s}, s) = 0 \& B \rangle \trianglerighteq_d P$ behaves like $\langle F(\dot{s}, s) = 0 \& B \rangle$ if it can terminate within d time units. Otherwise, after d (inclusive) time units, it behaves like P.
– $\langle F(\dot{s}, s) = 0 \& B \rangle \trianglerighteq [\!]_{i \in I}(io_i \to P_i)$ behaves like $\langle F(\dot{s}, s) = 0 \& B \rangle$ until a communication in the following context appears. Then it behaves like P_i immediately after the communication io_i occurs.
– P^* means that the execution of P can be repeated for arbitrarily finitely many times.
– $Sys_1 \parallel Sys_2$ behaves as if Sys_1 and Sys_2 are executed independently except that all communications along the common channels between Sys_1 and Sys_2 are to be synchronized. In order to guarantee that Sys_1 and Sys_2 have no

shared continuous nor discrete variables, and neither shared input nor output channels, we give the following syntactical constraints:

$$(\mathbf{VC}(Sys_1) \cap \mathbf{VC}(Sys_2)) = \emptyset,$$
$$(\mathbf{InChan}(Sys_1) \cap \mathbf{InChan}(Sys_2)) = \emptyset,$$
$$(\mathbf{OutChan}(Sys_1) \cap \mathbf{OutChan}(Sys_2)) = \emptyset,$$

where $\mathbf{VC}(Sys)$ stands for variables of Sys, $\mathbf{InChan}(Sys)$ for input channels of Sys and $\mathbf{OutChan}(Sys)$ for output channels of Sys.

Example: Plant Control (PLC)
A computer every d time units senses a plant, calculates the new control according to the sensed state and sends back to the plant. This can be modelled in HCSP as

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

where $contl(v)$ is an expression of v to stand for a calculation of the control parameter corresponding to v, which stores the sensed state.

3 Hoare Triple

The calculus is given in

$$\{PreH\}\ Sys\ \{PostH\},$$

which is similar to the Hoare triple but has $PreH$ and $PostH$ in Duration Calculus (DC) [13] to record pre-history and post-history of Sys.

DC is based on Interval Temporal Logic, and reasons about terms $\int S$, where S is a Boolean function over time (non-negative reals) and $\int S$ is the duration of state S within the reference time interval. We define

$$\begin{aligned} \ell &= \int 1, \\ \lceil S \rceil &= (\int S = \ell) \wedge (\ell > 0), \\ \lceil S \rceil^{<} &= \lceil S \rceil \vee (\ell = 0). \end{aligned}$$

Hence, for any given interval, ℓ is the length of the interval, and $\lceil S \rceil$ means that S holds (almost) everywhere in the interval and the interval is not a point one.

A *history formula* is a DC formula, or followed by the propositional letter C to stand for *Continuation*, or a disjunction of such formulas:

$$HF ::= A \mid A^\frown C \mid HF_1 \vee HF_2$$

where A is a DC formula without occurrence of C. $PreH$ and $PostH$ are history formulas.
Example: Stability of PLC

$$\{\lceil Controllable(s, u)\rceil^\frown C\}\ PLC\ \{(\ell > T) \Rightarrow ((\ell = T)^\frown \lceil \mid s - s_{target} \mid < \epsilon\rceil)\}$$

The pre-history requires that the initial state and control are *controllable* and the pre-history can be continued. The post-history concludes that after T time units the plant will be very close to the target (s_{target}).

In order to treat super-dense computation, we introduce N state to stand for *negligible* time. Therefore time is measured by $\int \neg N$.
Example: Stability of PLC becomes

$$\{\lceil Controllable(s,u) \wedge N \rceil^\frown C\}\ PLC\ \{(\int \neg N > T) \Rightarrow \\ ((\int \neg N = T)^\frown \lceil \mid s - s_{target} \mid < \epsilon \rceil)\}$$

4 Axioms and Rules

We introduce for each channel name c two shared states $c!$ and $c?$ to represent the *readiness* of output and input plus a shared variable c to store the message to be passed.

- **Monotonicity**
  If $\{PreH\} Sys \{PostH\}$, $(PreH' \Rightarrow PreH)$ and $(PostH \Rightarrow PostH')$, then

  $$\{PreH'\}\ Sys\ \{PostH'\}.$$

- **Disjunction**
  History formula can be restricted to disjunction of DC formulas with or without C as its last part. Correspondingly we can establish the following rule:

  $$\text{If } \{PreH_i\}\ Sys\ \{PostH_i\},\quad i = 1, ..., n,\\ \text{then } \{\textstyle\bigvee_{i=1}^n PreH_i\}\ Sys\ \{\bigvee_{i=1}^n PostH_i\}$$

  This rule can be generalized to the Existential one, such as

  $$\text{If } \{PreH\}\ Sys\ \{PostH\}\\ \text{then } \{\exists z.PreH\}\ Sys\ \{\exists z.PostH\}\\ \text{provided } z \notin \mathbf{VC}(Sys)$$

- **Skip**

  $$\{PreH\}\ \text{skip}\ \{PreH\}$$

  It means, skip does nothing and terminates immediately.
- **Assignment**
  If $(PreH[(\ell = 0)/C] \Rightarrow \top^\frown \lceil Pre[e/x] \rceil)$, then

  $$\{PreH\}\ x := e\ \{PreH[(\lceil Pre \wedge \neg \mathbf{Chan}(P) \wedge N \rceil^\frown C)/C]\}$$

  where we assume that Pre does not contain N nor channel variables, $\mathbf{Chan}(P)$ is $\{c? \mid c \in \mathbf{InChan}(P)\} \cup \{c! \mid c \in \mathbf{OutChan}(P)\}$, and, by $\neg\mathbf{Chan}(P)$, we mean the conjunction of $\neg c$, $c \in \mathbf{Chan}(P)$, assuming that the assignment statement is inside process P.

The hypothesis says that the last period of the pre-history (after ignoring C, i.e. C is replaced by $(\ell = 0)$) can conclude e satisfying Pre. Then the post-history can make sure that x satisfies Pre after the assignment, and no channels are ready for communication during the assignment. By N this rule also shows that an assignment consumes negligible time.

– **Wait**
If $(PreH[(\ell = 0)/C] \Rightarrow \top^\frown\lceil Pre\rceil)$, then

$$\{PreH\}\ \text{wait}\ d\ \{PreH[(\lceil Pre \wedge \neg\mathbf{Chan}(P)\rceil \wedge (\textstyle\int\neg N = d))^\frown C)/C]\}$$

where $d > 0$, Pre follows the assumption stated in the **Assignment** and so does in the followings. This rule specifies, wait d inherits the last state from $PreH$ and no channel is ready for communication during this waiting period (i.e. non-negligible time passes d).

– **Sequential Composition**

If $\{PreH_i\}P_i\{PostH_i\},\ i = 1, 2$, and $PostH_1 \Rightarrow PreH_2$, then

$$\{PreH_1\}\ P_1; P_2\ \{PostH_2\}$$

– **Conditional**

1. If $(PreH[(\ell = 0)/C] \Rightarrow \top^\frown\lceil B\rceil)$, then

$$\{PreH\}\ B \to P\ \{PostH\}$$

provided $\{PreH\}P\{PostH\}$.

2. If $(PreH[(\ell = 0)/C] \Rightarrow \top^\frown\lceil \neg B\rceil)$, then

$$\{PreH\}\ B \to P\ \{PreH\}$$

– **Internal Choice**

$$\begin{array}{l}\text{If } \{PreH\}\ P_i\ \{PostH_i\},\ i = 1, 2\\ \text{then } \{PreH\}\ P_1 \sqcup P_2\ \{PostH_1 \vee PostH_2\}\end{array}$$

– **Input**

$$\begin{array}{l}\text{If } \ PreH[(\ell = 0)/C]\ \Rightarrow\ \top^\frown\lceil Pre\rceil,\\ \text{then } \{PreH\}\ c?x\ \{PreH[\mathrm{In}(c, x)/C]\}\end{array}$$

where

$$\begin{array}{rcl}\mathrm{WaitIn}(c, x) & = & \lceil Pre \wedge c? \wedge \neg c! \wedge \neg(\mathbf{Chan}(P) \setminus \{c?\})\rceil\\ \mathrm{SynIn}(c, x) & = & \lceil(\exists x.Pre) \wedge c? \wedge c! \wedge (x = c) \wedge \neg(\mathbf{Chan}(P) \setminus \{c?\}) \wedge N\rceil^\frown\\ & & \lceil(\exists x.Pre) \wedge (x = c) \wedge \neg\mathbf{Chan}(P) \wedge N\rceil\\ \mathrm{In}(c, x) & = & \mathrm{WaitIn}(c, x)^{<\frown}\mathrm{SynIn}(c, x)^\frown C \vee \mathrm{WaitIn}(c, x)\end{array}$$

An input has to be firstly synchronized by an output that is described through WaitIn. Otherwise the input side will wait forever (i.e. the second disjunct of In cannot be continued). After the synchronization, a message

is input to x through c (i.e. $x = c$, as c stores the message), and the other variables do not change (i.e. $\exists x.Pre$). Here, we also assume, the message passing consumes negligible time and after it all channels become not ready for a negligible period to prevent multi-usage of a single message passing event.

– **Output**

$$\begin{array}{l}\text{If } \ PreH[(\ell = 0)/C] \ \Rightarrow \ \top^\frown\lceil Pre\rceil,\\ \text{then } \{PreH\}\ c!e\ \{PreH[\mathrm{Out}(c,e)/C]\}\end{array}$$

where

$$\begin{array}{rcl}\mathrm{WaitOut(c,e)} & = & \lceil Pre \wedge c! \wedge \neg c? \wedge \neg(\mathbf{Chan}(P) \setminus \{c!\})\rceil\\ \mathrm{SynOut}(c,e) & = & \lceil Pre \wedge c! \wedge c? \wedge (c = e) \wedge \neg(\mathbf{Chan}(P) \setminus \{c!\}) \wedge N\rceil^\frown\\ & & \lceil Pre \wedge (c = e) \wedge \neg\mathbf{Chan}(P) \wedge N\rceil\\ \mathrm{Out}(c,e) & = & \mathrm{WaitOut}(c,e)^{<\frown}\mathrm{SynOut}(c,e)^\frown C \vee \mathrm{WaitOut}(c,e)\end{array}$$

A symmetrical explanation can be given for the **Output**.

– **External Choice**

We use $c_1?x_1 \to P_1 \ [\!]\ c_2?x_2 \to P_2$ to explain this rule.

1. Let $(PreH[(\ell = 0)/C] \Rightarrow \top^\frown\lceil Pre\rceil)$.
2. Waiting Phase:

$$\mathrm{Wait} = \lceil Pre \bigwedge_{i=1}^{2}(c_i? \wedge \neg c_i!) \wedge \neg(\mathbf{Chan}(P) \setminus \{c_1?, c_2?\})\rceil$$

3. Synchronous Phase: for $i = 1, 2$

$$\begin{array}{rl}\mathrm{Syn}_i = & \lceil(\exists x_i.Pre) \wedge c_i! \wedge c_i? \wedge (x_i = c_i) \wedge \neg(\mathbf{Chan}(P) \setminus \{c_{\bar{i}}?\}) \wedge N\rceil^\frown\\ & \lceil(\exists x_i.Pre) \wedge (x_i = c_i) \wedge \neg\mathbf{Chan}(P) \wedge N\rceil\end{array}$$

where, in $c_{\bar{i}}$, $\bar{1} = 2$ and $\bar{2} = 1$.

4. If for $i = 1, 2$

$$\{PreH[(\mathrm{Wait}^{<\frown}\mathrm{Syn}_i^\frown C \ \vee \ \mathrm{Wait})/C]\}\ P_i\ \{PostH_i\}$$

then we can conclude

$$\{PreH\}\ c_1?x_1 \to P_1 \ [\!]\ c_2?x_2 \to P_2\ \{PostH_1 \vee PostH_2\}$$

– **Boundary Interruption**

Given a differential invariant Inv of $\langle F(\dot{s}, s) = 0 \& B\rangle$ with initial states satisfying $Init$

$$\begin{array}{l}\text{If } PreH[(\ell = 0)/C] \ \Rightarrow \ \top^\frown\lceil Init \wedge Pre\rceil, \text{ then}\\ \{PreH\}\ \langle F(\dot{s}, s) = 0 \& B\rangle\\ \quad \{PreH[((\lceil Inv \wedge Pre \wedge B \wedge \neg\mathbf{Chan}(P)\rceil^{<\frown}\\ \qquad \lceil Pre \wedge \mathbf{Close}(Inv) \wedge \mathbf{Close}(\neg B) \wedge \neg\mathbf{Chan}(P) \wedge N\rceil^\frown C)\\ \qquad \vee\lceil Inv \wedge Pre \wedge B \wedge \neg\mathbf{Chan}(P)\rceil)/C]\}\end{array}$$

where Pre does not contain s, and $\mathbf{Close}(G)$ is for the *closure* of G to include the boundary, e.g. $\mathbf{Close}(x < 2) = x \leq 2$.
During the evolution of s, Inv and B must hold and so does Pre for the variables other than s. However, when s stops, it will transit to the consecutive statement immediately (i.e. in negligible time). But, during the transition, $\neg B$ becomes true, or B reaches its boundary and $\mathbf{Close}(\neg B)$ becomes true (if B is closed). This can also argue for Inv.

– **Timeout Interruption**

$$\langle F(\dot{s}, s) = 0 \& B\rangle \trianglerighteq_d Q$$

can be semantically defined as

$$\langle F(\dot{s}, s) = 0,\ \dot{t} = 1 \& (B \wedge t < d)\rangle; ((t = d) \rightarrow Q)$$

with 0 as initial value of t.
For the **Boundary Interruption** rule, if we rewrite $\langle F(\dot{s}, s) = 0 \& B\rangle$ into $\langle F(\dot{s}, s) = 0,\ \dot{t} = 1 \& B\rangle$ and can generate a differential invariant which can deduce a range of t, say $Rg(t)$, then we can make sure that the duration of $\int \neg N$ for $\lceil Inv \wedge Pre \wedge B \wedge \neg \mathbf{Chan}(P)\rceil^{<}$ in the **Boundary Interruption** rule must satisfy $Rg(\int \neg N)$.

– **Communication Interruption**
The rule for $\langle F(\dot{s}, s) = 0 \& B\rangle \trianglerighteq [\!]_{i \in I}(io_i \rightarrow P_i)$ is a combination of the **Boundary Interruption** rule and the **External Choice** rule but quite complicated.
Here we use $\langle F(\dot{s}, s) = 0\rangle \trianglerighteq (c!s \rightarrow Q)$ to demonstrate its main idea. Assume Inv is a differential invariant of F for initial values $Init$, and

$$PreH[(\ell = 0)/C] \Rightarrow \top^\frown \lceil Init \wedge Pre\rceil,$$

where Pre does not contain s.

$$\begin{array}{l}\text{If } \{PreH[((\lceil Inv\rceil \wedge \text{WaitOut}(c, s))^{<\frown}\text{SynOut}(c, s))^\frown C \\ \qquad \vee(\lceil Inv\rceil \wedge \text{WaitOut}(c, s)))/C]\}\ Q\ \{PostH\}, \\ \text{then } \{PreH\}\ \langle F(\dot{s}, s) = 0\rangle \trianglerighteq (c!s \rightarrow Q)\ \{PostH\}\end{array}$$

Since B is $\top$, s can evolve forever unless an output over c occurs.

– **Repetition**:
We use the conventional history invariant as defined below

$$\begin{array}{l}\text{If } \{InvH\}\ P\ \{InvH\} \\ \text{then } \{InvH\}\ P^*\ \{InvH\}\end{array}$$

– **Parallel Composition**

$$\begin{array}{l}\text{If } \{PreH_i\}\ Sys_i\ \{PostH_i\} \\ \text{and } PostH_i[(\ell = 0)/C]\ \Rightarrow\ \top^\frown\lceil Post_i\rceil,\ i = 1, 2 \\ \text{then } \{\bigwedge_{i=1}^2 PreH_i\}\ Sys_1 \parallel Sys_2\ \{\bigwedge_{i=1}^2 PostH_i[\lceil Post_i\rceil/C]\}\end{array}$$

where $Post_i,\ i = 1, 2$ do not contain N and channel variables.
In order to avoid different length and occurrence of N state between parallel processes, we use $\lceil Post_i\rceil$ to fill up $PostH_i$, for $i = 1, 2$.

5 Discussion

1. The calculus can only prove safety property, although it introduces the concept of readiness. It is still a challenge to develop a calculus for liveness property.
2. To prove properties of HCSP processes, we have to find out appropriate differential invariants for various differential equations. Although [6] proposes an algorithm to establish polynomial invariants for polynomial differential equations, the complexity of the algorithm is terribly high. We are making efforts to establish nonlinear invariants with reasonable complexity.
3. In [5], the notation of HCSP includes $(P \trianglerighteq_d Q)$ and $(P \trianglerighteq [\!]_{i\in I}(io_i \to P_i))$, where P can be an arbitrary HCSP process. The history formulas of the calculus record all details of various HCSP processes. We believe that the calculus can be revised for [5].
4. Intuitively this calculus is sound. A rigorous proof of its soundness is to give HCSP another naive semantics and to prove consistency between the semantics and the calculus.

Acknowledgment This work has been partly supported by the 973 project with grant No. 2014CB340-700, and the projects from NSFC with grant No. 91118007 and 6110006.

References

1. Dimitar P. Guelev and Dang Van Hung. Prefix and Projection onto State in Duration Calculus. In *Proceedings of TPTS'02*, volume 65(6) of *ENTCS*, pp. 101-119, Elsvier Science, 2002.
2. Dimitar P. Guelev, Shuling Wang and Naijun Zhan. Hoare-style Reasoning about Hybrid CSP in the Duration Calculus. Technical report ISCAS-SKLCS-13-01, ISCAS, 2013.
3. Jifeng He. From CSP to hybrid systems. In *Proceedings of A Classical Mind*: Essays in Honour of C. A. R. Hoare, Prentice-Hall International Series In Computer Science, pp. 171-189. 1994.
4. C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.
5. Jiang Liu, Jidong Lv, Zhao Quan, Naijun Zhan, Hengjun Zhao, Chaochen Zhou and Liang Zou. A calculus for Hybrid CSP. In *Proceedings of APLAS'10*, LNCS 6461, pp.1-15, 2010.
6. Jiang Liu, Naijun Zhan and Hengjun Zhao. Computing semi-algebraic invariants for polynomial dynamical systems. In *Proceedings of EMSoft'11*, pp. 97-106, 2011.
7. Paritosh K. Pandya and Dang Van Hung. Duration Calculus of Weakly Monotonic Time. In *Proceedings of FTRTFT'98*, LNCS 1486, pp. 55-64, 1998.
8. André Platzer and Edmund M. Clark. Computing differential invariants of hybrid systems as fixedpoints. In *Proc. of CAV 2008*, LNCS 5123, pp. 176-189, 2008.
9. André Platzer and Jan-David Quesel. European train control system: A case study in formal verification. In *Proceedings of ICFEM '09*, pp. 246 - 265, 2009.
10. Bican Xia and Lu Yang. An algorithm for isolating the real solutions of semi-algebraic systems. *J. Symbolic Computation,* 34:461–477, 2002.

11. Lu Yang. Recent advances on determining the number of real roots of parametric polynomials. *J. Symbolic Computation*, 28:225–242, 1999.
12. Naijun Zhan, Shuling Wang and Hengjun Zhao. Formal Modelling, Analysis and Verification of Hybrid Systems, In *Unifying Theories of Programming and Formal Engineering Methods*, LNCS 8050, pp. 207-281, 2013.
13. Zhou Chaochen and Michael R. Hansen. Duration Calculus, A Formal Approach to Real-Time Systems. Springer, 2004.
14. Zhou Chaochen, Wang Ji, Anders P. Ravn. A formal description of hybrid systems. In *Proc. of Hybrid Systems'95*, LNCS 1066, pp. 511-530, 1995.
15. Liang Zou, Jidong Lv, Shuling Wang, Naijun Zhan, Tao Tang, Lei Yuan and Yu Liu. Verifying Chinese train control system under a combined scenario by theorem proving. In *Proceedings of VSTTE'13*, to appear in volume 8164 of the Lecture Notes in Computer Science.
16. Liang Zou, Naijun Zhan, Shuling Wang, Martin Fränzle and Shengchao Qin. Verifying Simulink Diagrams via a Hybrid Hoare Logic Prover. In *Proceedings of EMSoft'13*, pp. 1-10, 2013.