An Assume/Guarantee Based Compositional Calculus for Hybrid CSP

Shuling Wang[1], Naijun Zhan[1], and Dimitar Guelev[2]

[1] State Key Lab. of Comput. Sci., Institute of Software, Chinese Academy of Sciences
[2] Institute of Mathematics and Informatics, Bulgarian Academy of Sciences

Abstract. HCSP (Hybrid CSP) extends CSP to describe interacting continuous and discrete dynamics. The concurrency with synchronous communications, timing constructs, interrupts, differential equations, and so on, make the behavior of HCSP difficult to specify and verify. In this paper, we propose a Hoare-style calculus for reasoning about HCSP. The calculus includes Duration Calculus formulas to record process execution history and reason about real-time properties and continuous evolution, and dedicated predicate symbols to specify communication traces and readiness of process actions in a way which enables synchronisation to be handled compositionally by using assume/guarantee reasoning.

keywords: Hybrid Systems, Duration Calculus, Hoare Logic, HCSP, Compositionality, Assume/Guarantee

1 Introduction

Hybrid systems exhibit combinations of discrete jumps and continuous evolution. The applications of hybrid systems are dispersed everywhere in our modern life, e.g. industry automation and transport infrastructure incorporate many hybrid systems whose correct functioning is safety-critical. A number of abstract models and requirement specification languages have been proposed for the specification and verification of hybrid systems. The most popular model is *hybrid automata* [1, 10, 5], with real-time temporal logics [10, 11] interpreted on their behaviours as a specification language. However, hybrid automata are analogous to state machines, with little support for structured description, and for solving this problem, a number of formalisms have been proposed to facilitate modular descriptions of complex systems, e.g. Hybrid CSP [4, 20].

Hybrid CSP (HCSP) [4, 20] is a process algebra which extends CSP by real-time and continuous constructs, for instance differential equations to model continuous evolution. Being a process algebra, HCSP has standard means for constructing complex systems out of simpler ones, which facilitates compositionality. Our experience in formalising the Chinese Train Control System Level 3 (CTCS-3), has confirmed the applicability and scalability of HCSP. In this paper we propose a Hoare style calculus for reasoning about hybrid systems which are modelled in HCSP. The features of HCSP which are handled by the logic include communication, timing constructs, interrupts and continuous evolution governed by differential equations. The proposed calculus includes Duration Calculus [19] formulas to record execution history. A calculus for HCSP with similar features was proposed in a previous work [8][3], but compositionality, which is the main contribution of this work, was not achieved.

Compositionality in reasoning about properties of HCSP is a challenge because of continuous evolution and communication dependencies between processes, much like in other models of real time concurrency. Our approach to obtain the compositionality is inspired by the work on CSP without timing [6, 15, 17, 13]. To facilitate the reasoning about individual parallel components, we extend state expressions in *DC* history formulas by introducing predicates to specify communication readiness. Furthermore, to achieve the compositional reasoning of a compound construct such as sequential composition and parallel composition, we adopt assume/guarantee mechanism. Firstly, define the specification of each constituent independently, including a precondition, an assumption to specify conditions of the environment in which the constituent is executed, a postcondition and a guarantee, then the specification of the construct can be deduced from specifications of its constituents directly, for instances, sequential composition by chop, and parallel composition by conjunction, etc.

Related Work Hybrid automata [1, 5] and the related logics [10, 5, 11] were already mentioned in the introduction. Another approach is Differential Dynamic Logic proposed in [14] for the deductive verification of hybrid programs. However, the hybrid programs considered there have limited functionality. Communication, parallelism and interrupts are not handled.

By extending Hoare Logic, a compositional proof system for real-time concurrent systems with communications along asynchronous channels is presented in [7]. However, it assumes that each communication has a fixed non-zero duration. Instead, we adopt super-dense computation [10] to assume computer computation consuming zero time compared to continuous evolution, which is a very comfortable abstract computation model for hybrid systems and has been successfully applied to the formal design of hybrid systems in practice. Thus, at one time, there may be multiple communications being taken. We find that the compositionality of reasoning becomes more difficult in the super-dense computation model. For logic compositionality, assume/guarantee reasoning has been studied for communication-based concurrency in CSP without timing in [12, 13, 21].

Structure of the paper Section 2 gives a brief introduction to HCSP, and then we introduce DC formulas and define trace and readiness predicates for HCSP in Section 3. The assume/guarantee compositional calculus for HCSP is presented in Section 4. We draw a conclusion and discuss future work in Section 5.

[3] It admits interrupting discrete operations. Therefore its history formulas have to keep the records of possible change of any discrete variable. Hence, the Monotonicity Rule of [8] is not correct for history formulas, and must be weakened to a conservative one.

2 Hybrid CSP

Hybrid CSP [4, 20] is a formal language for describing hybrid systems. It is an extension of CSP by adding timing constructs, interruptions, and differential equations for representing continuous evolution. Interactions among processes are described solely by communications. Process variables are local to their respective sequential components. We write **Var** and **Chan** for the sets of the variables and the channel names occurring in the considered process, respectively. The syntax of a subset of HCSP is given as follows:

$$\begin{array}{ll} P, Q & ::= \mathrm{skip} \mid x := e \mid \mathrm{wait}\ d \mid ch?x \mid ch!e \mid P; Q \mid B \to P \mid P \sqcup Q \\ & \quad \mid [\!]_{i \in I}(io_i \to Q_i) \mid P \| Q \mid P^* \mid \langle \mathcal{F}(\dot{s}, s) = 0 \& B \rangle \\ & \quad \mid \langle \mathcal{F}(\dot{s}, s) = 0 \& B \rangle \unrhd_d Q \mid \langle \mathcal{F}(\dot{s}, s) = 0 \& B \rangle \unrhd [\!]_{i \in I}(io_i \to Q_i) \\ io_i & ::= ch?x \mid ch!e \end{array}$$

Here P, Q, x, s and ch stand for HCSP processes, (vectors of) real variables, and channel names, respectively. B and e are Boolean and arithmetic expressions and d is a non-negative real constant. The intended meaning of the individual constructs is as follows:

- skip terminates immediately having no effect on variables.
- $x := e$ assigns the value of expression e to x and then terminates.
- wait d will keep idle for d time units keeping variables unchanged.
- $ch?x$ receives a value along channel ch and assigns it to x.
- $ch!e$ sends the value of e along channel ch. A communication takes place when both the sending and the receiving parties are ready, and may cause one side to wait.
- The sequential composition $P; Q$ behaves as P first, and if it terminates, as Q afterwards.
- The alternative $B \to P$ behaves as P if B is true, otherwise it terminates immediately.
- $P \sqcup Q$ denotes internal choice. It behaves as either P or Q, and the choice is made by the process.
- $[\!]_{i \in I}(io_i \to Q_i)$ denotes communication controlled external choice. I is supposed to be finite. As soon as one of the communications io_i takes place, the process continues as the respective guarded Q_i.
- $P \| Q$ behaves as if P and Q run independently except that all communications along the common channels connecting P and Q are to be synchronized. The processes P and Q in parallel can neither share variables, nor input or output channels.
- The repetition P^* executes P for some finite number of times.
- $\langle \mathcal{F}(\dot{s}, s) = 0 \& B \rangle$ is the continuous evolution statement (hereafter shortly *continuous*). It forces the vector s of real variables to obey the differential equations $\mathcal{F}$ as long as the boolean expression B, which defines the *domain of* s, holds, and terminates when B turns false.
- $\langle \mathcal{F}(\dot{s}, s) = 0 \& B \rangle \unrhd_d Q$ behaves like $\langle \mathcal{F}(\dot{s}, s) = 0 \& B \rangle$, if that continuous terminates before d time units. Otherwise, after d time units of evolution according to $\mathcal{F}$, it moves on to execute Q.

– $\langle \mathcal{F}(\dot{s}, s) = 0\&B\rangle \trianglerighteq []_{i\in I}(io_i \to Q_i)$ behaves like $\langle \mathcal{F}(\dot{s}, s) = 0\&B\rangle$, except that the continuous evolution is preempted as soon as one of the communications io_i is taken place. That is followed by the respective Q_i. Notice that, if a non-B state is reached before a communication from among $\{io_i\}_I$ occurs, then the process terminates without communicating.

For reasoning about communication behaviour of HCSP, several auxiliary variables that never occur in any process need to be introduced: The system variable *now* records the current time of process execution, and the variable tr records the *timed trace* of a process accumulated during its execution. A *timed trace* (abbreviated as trace below) h can be either empty sequence, or $\langle ch.e, t\rangle$ denoting one occurrence of a communication $\langle ch, e\rangle$ at time t, or composed from existing traces by concatenation $\cdot$, non-deterministic choice $+$, and Kleene star *. We use r to denote the corresponding *channel sequence* of h, and define a function $\mathbb{C}(tr)$ to return the channel sequence of the trace recorded by tr. The formal definitions are presented as follows.

$$\begin{aligned} h &::= \epsilon \mid \langle ch.e, t\rangle \mid h_1 \cdot h_2 \mid h_1 + h_2 \mid h^* \\ r &::= \epsilon \mid ch \mid r_1 \cdot r_2 \mid r_1 + r_2 \mid r^* \end{aligned}$$

There are some properties held for the non-deterministic choice, including the distributivity of it over concatenation, e.g., $(h_1 + h_2) \cdot h = h_1 \cdot h + h_2 \cdot h$, and r as well; and the equivalent conversion from it to disjunction in assertions, e.g., $tr = h_1 + h_2$ iff $tr = h_1 \vee tr = h_2$.

Formal semantics of HCSP has been considered in different paradigms. For example, an algebraic semantics was given in [4], while a DC-based denotational semantics was given in [20]. In the full version of this paper [16], a formal operational semantics was given in the Plotkin's style, i.e., each construct of HCSP is interpreted as a transition relation over configurations composed of a process and a pair of states (for process and environment respectively), and the semantics is defined by a set of transition rules. For space limitation, we omit this part here.

3 History Formulas

Duration calculus (DC) [19] is an interval-based logic for specifying and reasoning about real-time systems. We will use DC formulas to describe the execution history of HCSP processes. However, in order to specify communications, we need to augment the state expressions of DC to include assertions related to communication readiness.

The syntax of the subset of *DC* we need is described in terms of *state expressions* S, which are assertions about variables, and *history formulas* HF as follows:

$$\begin{aligned} \theta \;&::= c \mid x \mid f^n(\theta_1, ..., \theta_n) \\ S \;&::= \bot \mid R^m(\theta_1, ..., \theta_m) \mid r.ch? \mid r.ch! \mid \mathcal{T}(P) \mid \neg S \mid S \vee S \\ HF &::= \bot \mid \ell \texttt{ rel } c \mid \lceil S\rceil^- \mid \lceil S\rceil^0 \mid HF^* \mid HF^\frown HF \mid HF \wedge HF \mid HF \vee HF \end{aligned}$$

Here θ stands for a *term*. c denotes a constant, x a process variable, and f is an n-ary arithmetic function (n as well as the following m are non-negative integers for representing arities of functions).

In the syntax of state expressions S, $\perp$ stands for false ($\top$ for true in contrary), and R is an m-ary truth-valued function on terms. In order to model the readiness of channel ch for performing communication, we introduce two Boolean variables $r.ch?, r.ch!$, with a channel sequence r (as defined in last section) as prefix, to describe that $ch?$ or $ch!$ becomes ready, and before that, the communication history along r has occurred. $\mathcal{T}(P)$ is a terminal predicate, representing that P terminates.

In the syntax of formulas HF, ℓ is a temporal variable standing for the length of the considered interval. `rel` is a relation in the set $\{<, >, =\}$. In the following, we always use $Rg(\ell)$ to denote an interval formula of ℓ, i.e., a history formula containing ℓ and constants. $\lceil S\rceil^{-}$ means that S holds *everywhere* in the right-open interval[4], and $\lceil S\rceil^{0}$ means that S holds at an isolated point. In the rest of the paper, we define the abbreviation $\lceil S\rceil \overset{\text{def}}{=} \lceil S\rceil^{-}{}^\frown\lceil S\rceil^{0}$, meaning that S holds everywhere over an interval. HF^* denotes iteration of history formulas. See, e.g., [2, 3] on iteration and some other relevant DC operators. In $HF_1{}^\frown HF_2$, ${}^\frown$ chops an interval into two consecutive sub-intervals, over which HF_1 and HF_2 hold respectively.

The semantics of terms, state expressions and history formulas are interpreted over process states. For the full semantics, readers are referred to [16].

Axioms All the axioms of DC are applied here. Besides, we need to introduce an axiom for readiness, for translating non-deterministic choice of terms equivalently into disjunction of state expressions,

$$(r_1 + r_2).ch? = r_1.ch? \vee r_2.ch?$$

4 Specification and Inference Rules

Unlike assertions defined in our previous work [8], each specification of Hybrid Hoare Logic (HHL) here consists of five parts, i.e. pre- and post-conditions, process, assumption and guarantee, with the form

$$\{S;\ A\}P\{R;\ G\}$$

P is an HCSP process to be verified. S and R are pre- and post-conditions which are state-trace assertions (that do not refer to the readiness $r.ch?, r.ch!$) about variables at the start and termination of the execution of P, respectively. A and G are both history formulas. Assumption A specifies the readiness of communications that the environment offers to P, while guarantee G specifies the execution history of P, when P runs under an environment satisfying A.

[4] The original DC defines the almost everywhere formula, written by $[\![S]\!]$. Here we use different variants of it.

Intuitively, a specification $\{S;\ A\}P\{R;\ G\}$ is valid, iff for any execution of P starting from a state satisfying S, if it terminates, and the environment under which P runs satisfies A throughout its execution, then the final state satisfies R, and G holds throughout the execution of P.

In the following, we will briefly introduce axioms and inference rules of HHL, detailed explanation can be referred to [16]. First we give general rules that are applicable to all HCSP statements, and then the rules for each HCSP construct.

Consequence Rule The consequence rule is defined as usual,

$$\frac{\{S;\ A\}P\{R;\ G\} \quad S' \Rightarrow S \quad R \Rightarrow R' \quad A' \Rightarrow A \quad G \Rightarrow G'}{\{S';\ A'\}P\{R';\ G'\}}$$

Non-readiness Rule This rule is closely related to the fact that each channel end is owned solely by one sequential context in HCSP. The communication actions that are sequential to P but not belonging to P are not ready when P is executing. For every process P, we assume that the processes that are composed with P in sequence have channel ends from $\mathcal{C}_S$. Let $\mathcal{C}_P$ be the set of channel ends of P, and $\mathcal{C}_N$ be $\mathcal{C}_S \setminus \mathcal{C}_P$. We then have the following rule describing the non-readiness of communication actions in $\mathcal{C}_N$ during the execution of P (the terminating point exclusive). The hypothesis $S \Rightarrow \mathbb{C}(tr) = r$ indicates that the processes previous to P have accumulated trace along channel sequence r.

$$\frac{\{S;\ A\}P\{R;\ G\} \quad S \Rightarrow \mathbb{C}(tr) = r}{\{S;\ A\}P\{R;\ G \wedge \lceil \bigwedge_{a \in \mathcal{C}_N} (\neg r.a) \rceil^-\}}$$

There are other general rules standard for classical predicate logic, similar to the ones presented in [14]. We will not list them here.

The rules for skip and assignment are straightforward. Both are internal actions, having no dependence from the environment, and take no time.

Skip

$$\{S;\ \top\} \text{ skip } \{S;\ \ell = 0\}$$

Assignment

$$\{S[e/x];\ \top\}\ x := e\ \{S;\ \ell = 0\}$$

Input and output The input rule is:

$$\frac{S \Rightarrow \mathbb{C}(tr) = r \quad S[o/now] \wedge Rg(t) \wedge now = o + t \Rightarrow \forall a. R[a/x, tr'/tr]}{\{S;\ (Rg(\ell) \wedge \lceil \neg(r.ch!) \rceil^-)^\frown \lceil r.ch! \rceil^0\}\ ch?x\ \{R;\ Rg(\ell) \wedge \lceil r.ch? \rceil\}}$$

where $Rg(\ell)$ is an interval formula of l, and t is a fresh logical variable, $tr' = tr \cdot \langle ch.a, now \rangle$. The assumption indicates that the partner side $ch!$ is not ready until $Rg(\ell)$ time units, and under this assumption, $ch?$ will keep waiting for the same duration. Whenever both parties get ready, the communication occurs immediately. The system clock now then goes ahead t with range $Rg(t)$, which is the waiting time, and a value is transmitted along ch and assigned to x, and tr is increased by one communication pair $\langle ch.a, now\rangle$, as indicated by the second hypothesis.

The rule for output can be defined similarly.

$$\frac{S \Rightarrow \mathbb{C}(tr) = r \quad S[o/now] \wedge Rg(t) \wedge now = o + t \Rightarrow R[tr'/tr]}{\{S;\ (Rg(\ell) \wedge \lceil \neg(r.ch?)\rceil^{-})^\frown \lceil r.ch?\rceil^{0}\}\ ch!e\ \{R;\ Rg(\ell) \wedge \lceil r.ch!\rceil\}}$$

where $tr' = tr \cdot \langle ch.e, now\rangle$.

Continuous For reasoning about the continuous, the notion of differential invariants is necessary, which is quite similar to reasoning about properties of loops using invariant in the classical Hoare logic. A differential invariant of a differential equation $\langle \mathcal{F}(\dot{s}, s) = 0 \& B\rangle$ for given initial values of s is a first order formula of s, which is satisfied by the initial values and kept satisfied during the continuous evolution following $\mathcal{F}$ within the domain defined by B. More details about differential invariants can be found in [9]. Moreover, as discussed in [8], the execution time of $\langle \mathcal{F}(\dot{s}, s) = 0 \& B\rangle$, can be counted by introducing a fresh local clock with initial value 0, that is, the value of t at the terminating point of $\langle \mathcal{F}(\dot{s}, s) = 0; \dot{t} = 1 \& B\rangle$.

Given a differential invariant Inv and the execution time $Rg(t)$ of $\langle \mathcal{F}(\dot{s}, s) = 0 \& B\rangle$ with initial values satisfying $Init$, we have the following rule:

$$\frac{S[o/now] \wedge Rg(t) \wedge now = o + t \Rightarrow R}{\begin{array}{c}\{Init \wedge S; \top\}\ \langle \mathcal{F}(\dot{s}, s) = 0 \& B\rangle\ \{R \wedge \mathbf{close}(Inv) \wedge \mathbf{close}(\neg B); \\ (l = 0 \vee \lceil S \wedge Inv \wedge B\rceil^{-}) \wedge Rg(\ell)\}\end{array}}$$

where S, R do not contain s. The notation $\mathbf{close}(Inv)$ stands for the closure of Inv, e.g, $s <= 5$ is closure of $s < 5$, to deal with the case when Inv does not hold at the escaping boundary. $\mathbf{close}(\neg B)$ similarly. Obviously, both closures of Inv and $\neg B$ hold when the continuous terminates, as shown in the post-condition. The continuous evolves for $Rg(t)$ time units and then terminates, and as a consequence, now is added by t with range $Rg(t)$ also. $l = 0$ in the history is to record the behavior that the initial value of s fails to satisfy B, then the continuous terminates immediately.

Conditional The statement $B \to P$ behaves like P when B is true, otherwise terminates immediately.

$$\frac{S \Rightarrow B \quad \{S;\ A\}P\{R;\ G\}}{\{S;\ A\}B \to P\{R;\ G\}} \text{ and } \frac{S \Rightarrow \neg B}{\{S;\ \top\}B \to P\{S;\ \ell = 0\}}$$

Sequential composition

$$\frac{\{S_1; A_1\}\ P_1\ \{R_1; G_1\} \quad \{S_2; A_2\}\ P_2\ \{R_2; G_2\} \quad R_1 \Rightarrow S_2}{\{S_1; A_1^\frown \lceil \mathcal{T}(P_1)\rceil^{0\frown} A_2\}\ P_1; P_2\ \{R_2; G_1^\frown G_2\}}$$

For $P_1; P_2$, the first component P_1 ends in a state satisfying post-condition R_1, from which the second P_2 starts to execute. Moreover, under the assumptions

A_1 and A_2, the executions of P_1 and P_2 guarantee G_1 and G_2 respectively. The assumption and guarantee of overall sequential composition can then be defined by chopping the ones of its components together. However, the assumption of P_1 should not assume anything about the environment of P_2, and vice versa. This is why we add the terminal predicate $\mathcal{T}(P_1)$ in between A_1 and A_2, indicating that the environment of P_1 terminates simultaneously as P_1. More discussions on the predicate will be given in the discussion section.

Parallel composition In order to define the rule, we need to introduce some notations first. Given a timed trace h and a channel set C, we denote by $h|_C$ the projection of h onto C, which removes all timed communications not along C from h. Similarly, we define the projection of a readiness variable $r.ch?$ (resp. $r.ch!$) onto C, denoted by $r.ch?|_C$ (resp. $r.ch!|_C$) as when $ch \notin C$ then $\top$, otherwise $r|_C.ch?$ (resp. $r|_C.ch!$), where $r|_C$ stands for the resulting channel sequence after filtering all occurrences of channels not in C from r. Accordingly, we define the projection of HF onto C, denoted by $HF|_C$ by replacing each occurrence of all readiness variables rv by $rv|_C$. Given two timed traces h_1 and h_2 and a set of channels C, we say h_1 and h_2 are *compatible* w.r.t. C, iff $h_1|_C = h_2|_C$, i.e., they have the same projection onto C. Given two timed traces h_1 and h_2 that are compatible w.r.t. C, we define the *alphabetized parallel* of h_1 and h_2 over C, denoted by $h_1 \underset{C}{\|} h_2$, defined by structural induction in Fig. 1. In the definition, we use **Undef** to represent that the resulting trace is undefined. Obviously, the alphabetized parallel $\underset{C}{\|}$ of two compatible traces w.r.t. C will always be well defined.

$$
\begin{array}{ll}
h_1 \underset{C}{\|} \epsilon & \overset{\text{def}}{=} \begin{cases} h_1 & \text{if } h_1|_C = \epsilon \\ \textbf{Undef} & \text{otherwise} \end{cases} \\
\langle ch_1.a, t_1\rangle \cdot h_1' \underset{C}{\|} \langle ch_2.b, t_2\rangle \cdot h_2' & \overset{\text{def}}{=} \begin{cases} \langle ch_1.a, t_1\rangle \cdot (h_1' \underset{C}{\|} h_2') \\ \quad \text{if } ch_1 = ch_2 \in C,\ a = b \text{ and } t_1 = t_2 \\ \langle ch_1.a, t_1\rangle \cdot (h_1' \underset{C}{\|} (\langle ch_2.b, t_2\rangle \cdot h_2')) \\ \ +\langle ch_2.b, t_2\rangle \cdot ((\langle ch_1.a, t_1\rangle \cdot h_1') \underset{C}{\|} h_2') \\ \quad \text{otherwise if } ch_1, ch_2 \notin C \text{ and } t_1 = t_2 \\ \langle ch_1.a, t_1\rangle \cdot (h_1' \underset{C}{\|} (\langle ch_2.b, t_2\rangle \cdot h_2')) \\ \quad \text{otherwise if } ch_1 \notin C, \text{ and } t_1 \leq t_2 \\ \langle ch_2.b, t_2\rangle \cdot ((\langle ch_1.a, t_1\rangle \cdot h_1') \underset{C}{\|} h_2') \\ \quad \text{otherwise if } ch_2 \notin C, \text{ and } t_2 \leq t_1 \\ \textbf{Undef} \ \text{otherwise} \end{cases} \\
(h_1' + h_2') \underset{C}{\|} (h_1'' + h_2'') & \overset{\text{def}}{=} \Sigma_{i,j=1,2}(h_i' \underset{C}{\|} h_j'')
\end{array}
$$

Fig. 1. Alphabetized parallel of timed traces

Now we define the rule for the case when P_1 and P_2 terminate simultaneously. It can be generalised easily for other cases. Let $C_i = \mathbf{Chan}(P_i)$ for $i = 1, 2$, and $C = C_1 \cap C_2$. The parallel rule is given as follows:

$$\frac{\begin{array}{c}\{S_1; A_1\}\ P_1\ \{R_1; G_1\} \quad \{S_2; A_2\}\ P_2\ \{R_2; G_2\} \\ G_1|_C \Rightarrow A_2|_C \quad A|_{C_2 \backslash C} \Rightarrow A_2|_{C_2 \backslash C} \quad G_2|_C \Rightarrow A_1|_C \quad A|_{C_1 \backslash C} \Rightarrow A_1|_{C_1 \backslash C}\end{array}}{\{S_1 \wedge S_2; A\}\ P_1 \| P_2\ \{\mathsf{comp}(R_1, R_2); G_1 \wedge G_2\}}$$

where $\mathsf{comp}(R_1, R_2)$ is defined as follows:

$$\mathsf{comp}(R_1, R_2) \overset{\text{def}}{=} R_1[tr_1/tr] \wedge R_2[tr_2/tr] \wedge tr_1|_C = tr_2|_C \wedge tr = tr_1 \underset{C}{\|} tr_2$$

It indicates that, because of synchronous communication, P_1 and P_2 will produce compatible traces along the common channel set C; moreover, the final trace tr is the alphabetized parallel of the traces of P_1 and P_2 over C. The parallel rule says that, we need to check the compatibility, i.e., the assumption of each process in the parallel composition must be fulfilled by its environment, including the other process in parallel with it and the external environment separately.

External and internal choice The external choice depends on external environment totally, i.e., whose partner comes earlier, who is chosen to execute. We just present the rule for the case of $ch?x \rightarrow P \talloblong dh!e \rightarrow Q$, which can easily generalized to the general case. The first rule describes the case when the partner of $ch?$ gets ready before the one of $dh!$, described by A.

$$\frac{\begin{array}{c}S \Rightarrow \mathbb{C}(tr) = r \quad A \Rightarrow \lceil \neg(r.ch! \wedge r.dh?) \rceil^{-\frown} \lceil r.ch! \wedge \neg(r.dh?) \rceil^{0\frown} \top \\ \{S;\ A\}\ ch?x; P\ \{R;\ G\}\end{array}}{\{S;\ A\}\ ch?x \rightarrow P \talloblong dh!e \rightarrow Q\ \{R;\ G\}}$$

The symmetric case when the partner of $dh!$ gets ready before the one of $ch?$ can be defined similarly. However, whenever the partners of $ch?$ and $dh!$ get ready simultaneously, the external choice becomes non-deterministic choice, and therefore, one of the alternatives is chosen by the process randomly, as defined by the following rule:

$$\frac{\begin{array}{c}S \Rightarrow \mathbb{C}(tr) = r \quad A \Rightarrow \lceil \neg(r.ch! \wedge r.dh?) \rceil^{-\frown} \lceil r.ch! \wedge r.dh? \rceil^{0\frown} \top \\ \{S;\ A\}\ ch?x \rightarrow P\ \{R_1;\ G_1\} \quad \{S;\ A\}\ dh!e \rightarrow Q\ \{R_2;\ G_2\}\end{array}}{\{S;\ A\}\ ch?x \rightarrow P \talloblong dh!e \rightarrow Q\ \{R_1 \vee R_2;\ G_1 \vee G_2\}}$$

In contrast to external choice, the internal choice $P_1 \sqcup P_2$ depends on the process totally, and the choice is made randomly by the process itself. To prevent deadlock, the environment must provide the assumptions required by both alternatives, and the internal choice guarantees the behavior of one of them.

$$\frac{\{S;\ A\}\ P_i\ \{R_i;\ G_i\} \quad \text{for } i = 1, 2}{\{S;\ A\}\ P_1 \sqcup P_2\ \{R_1 \vee R_2;\ G_1 \vee G_2\}}$$

Interrupt by communication For process $\langle \mathcal{F}(\dot{s}, s) = 0 \& B\rangle \trianglerighteq (ch?x \rightarrow Q)$, the continuous will be executed first, and interrupted once the communication along ch happens, and Q will be executed afterwards. However, if the communication does not happen before the domain restriction B becomes false, the process will not wait for the communication and terminate immediately.

The first rule for the case when the communication occurs before the continuous terminates, as indicated by $Rg'(\ell) \Rightarrow Rg(\ell)^\frown \top$.

$$\frac{\begin{array}{c} S \Rightarrow \mathbb{C}(tr) = r \quad \{S \wedge Init; \top\}\langle \mathcal{F}(\dot{s}, s) = 0 \& B\rangle\{R;\ G \wedge Rg'(\ell)\} \\ A \Rightarrow Rg(\ell) \wedge \lceil \neg(r.ch!)\rceil^{-\frown}\lceil r.ch!\rceil^{0\frown}\top \quad Rg'(\ell) \Rightarrow Rg(l)^\frown\top \\ G \Rightarrow \lceil Inv\rceil^* \quad \{S \wedge Inv;\ A\}\ ch?x; Q\ \{R';\ G'\} \end{array}}{\{S \wedge Init;\ A\}\ \langle \mathcal{F}(\dot{s}, s) = 0 \& B\rangle \trianglerighteq (ch?x \rightarrow Q)\ \{R';\ ((Rg(\ell) \wedge \lceil Inv\rceil^-)^\frown\top) \wedge G'\}}$$

where S does not contain s.

The second rule for the other case when the communication does not happen until the continuous terminates, as indicated by $Rg(l) \Rightarrow Rg'(\ell)^\frown(l > 0)$.

$$\frac{\begin{array}{c} S \Rightarrow \mathbb{C}(tr) = r \quad \{S \wedge Init;\ \top\}\ \langle \mathcal{F}(\dot{s}, s) = 0 \& B\rangle\ \{R;\ G \wedge Rg'(\ell)\} \\ A \Rightarrow Rg(\ell) \wedge \lceil \neg(r.ch!)\rceil^\frown\top \quad Rg(\ell) \Rightarrow Rg'(\ell)^\frown(\ell > 0) \end{array}}{\{S \wedge Init;\ A\}\ \langle \mathcal{F}(\dot{s}, s) = 0 \& B\rangle \trianglerighteq (ch?x \rightarrow Q)\ \{R;\ G\}}$$

Repetition Similar to the classical Hoare logic, we first need to find an invariant S' that holds before and after the execution of the process P. Second, the assumption and guarantee of P^* can be defined as the iteration of the ones of P. Similar to sequential composition, for each iteration of P, the environment terminates simultaneously as P does, as guaranteed by $\lceil \mathcal{T}(P)\rceil^0$ in the assumption.

$$\frac{S \Rightarrow S' \quad \{S';\ A\}P\{S';\ G\}}{\{S;\ (A^\frown\lceil \mathcal{T}(P)\rceil^0)^*\}\ P^*\ \{S';\ G^*\}}$$

We do not define the rules for wait and timeout constructs here, as both of them are not primitive, and can be defined by the continuous and other constructs.

5 Discussions, Conclusion and Future Work

Total Correctness vs Partial Correctness

In this paper, we assume that each HCSP process terminates in a finite time, as we adopt the classical DC to specify assumptions and guarantees, with which infinite behaviour of a system cannot be defined. So, we just discuss partial correctness here. In [18], Duration Calculus is extended with infinite intervals, which can be used to distinguish termination and non-termination simply.

On the other hand, the predicate $\mathcal{T}$ has been introduced for representing the termination of a process in the calculus. It is used for specifying that a process and its environment terminate at the same time. We believe the predicate can be used to distinguish termination and non-termination as well, but this will complicate the inference rules. In addition, the proof system presented here is incomplete as we at least omit several rules for reasoning about the predicate $\mathcal{T}$. We will leave this issue as one future work.

Conclusion and Future Work

In this paper, we present a compositional calculus for specifying and verifying hybrid systems. The language for modeling hybrid systems is a subset of HCSP, by using which we have modelled the train movement scenarios of CTCS-3, thus show the modelling expressiveness of HCSP. By introducing DC formulas into Hoare Logic to record the execution history of HCSP, the calculus can specify real-time and continuous properties of Hybrid systems. By introducing predicates for describing communication traces and readiness, based on assume/guarantee method, the calculus can specify time and communication synchronisation between parallel processes compositionally. However, the calculus is a little complicated, and we will try to simplify it as one future work.

To establish deadlock freedom of a process, it is necessary to record information of readiness of different actions during the execution of the process. The predicates introduced for specifying readiness of communication actions can provide a basis. Moreover, we will try to apply this calculus to prove practical hybrid systems, e.g., the movement scenarios of CTCS-3.

Acknowledgment The authors would like to thank Prof. Chaochen Zhou for his insightful suggestions and comments on this paper. This work has been supported by NSFC projects 91118007, 60970031 and 61100061.

References

1. R. Alur, C. Courcoubetis, T. A. Henzinger, and P. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid Systems. LNCS, Volume 736/1993*, pages 209–229. Springer-Verlag, 1992.
2. D. P. Guelev and V. H. Dang. Prefix and projection onto state in duration calculus. *Proceedings of the ETAPS workshop Theory and Practice of Timed Systems (TPTS'02), ENTCS*, 65(6):101–119, 2002.
3. D. P. Guelev and V. H. Dang. On the completeness and decidability of duration calculus with iteration. *Theoretical Computer Science*, 337:278–304, 2005.
4. J. He. *From CSP to hybrid systems*, pages 171–189. Prentice Hall International (UK) Ltd., 1994.
5. T. A. Henzinger. The theory of hybrid automata. In *Proc. 11th Annu. IEEE Symp. Logic Comput. Sci. (LICS'96)*, pages 278–292, 1996.
6. C. A. R. Hoare. A calculus of total correctness for communicating processes. *Sci. Comput. Program.*
7. J. Hooman. Extending Hoare logic to real-time. *Formal Aspects of Computing*, 6:801–825, 1994.

8. J. Liu, J. Lv, Z. Quan, N. Zhan, H. Zhao, C. Zhou, and L. Zou. A calculus for hybrid CSP. In *Programming Languages and Systems, LNCS*, volume 6461/2010. Springer, 2010.
9. J. Liu, N. Zhan, and H. Zhao. Computing semi-algebraic invariants for polynomial dynamical systems. In *EMSOFT '11*, pages 97–106. ACM, 2011.
10. Z. Manna and A. Pnueli. Verifying hybrid systems. In *Grossman et al*, pages 4–35. Springer-Verlag, 1993.
11. Z. Manna and H. Sipma. Deductive verification of hybrid systems using STeP. In *HSCC*, volume 1386 of *LNCS*, pages 305–318, 1998.
12. J. Misra and M. Chandy. Proofs of networks of processes. In *IEEE SE*, volume 7, pages 417–426, 1981.
13. P.K. Pandya and M. Joseph. P-A logic - a compositional proof system for distributed programs. In *Distributed Computing*, volume 5, pages 37–54, 1991.
14. A. Platzer. Differential dynamic logic for hybrid systems. *J. Autom. Reasoning*, 41(2):143–189, 2008.
15. N. Soundararajan. Axiomatic semantics of communicating sequential processes. *ACM Transactions on Programming Languages and Systems*, 6:647–662, 1984.
16. S. Wang, N. Zhan, and D. Guelev. An assume/guarantee based compositional calculus for Hybrid CSP and its soundness. Technical report, State Key Lab. of Computer Science, ISCAS, 2011.
17. C. Zhou. Specifying communicating systems with temporal logic. In *Temporal Logic in Specification*, volume 398 of *LNCS*, pages 304–323. Springer, 1987.
18. C. Zhou, V. Dang, and X. Li. A duration calculus with infinite intervals. In *Fund. of Comput. Theory*, volume 965 of *LNCS*, pages 16–41. Springer, 1995.
19. C. Zhou and M. R. Hansen. *Duration Calculus: A Formal Approach to Real-Time Systems.* Series: Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2004.
20. C. Zhou, J. Wang, and A. P. Ravn. A formal description of hybrid systems. In *Proceedings of the DIMACS/SYCON workshop on Hybrid systems III : verification and control*, pages 511–530. Springer-Verlag, 1996.
21. J. Zwiers, A. de Bruin, and W.-P. de Roever. A proof system for partial correctness of dynamic networks of processes. In *Proc. of the Conference on Logics of Programs, LNCS 164*, 1984.