# Denial-of-Service Security Attack in the Continuous-Time World

Shuling Wang, Flemming Nielson, Hanne Nielson



# Denial-of-Service Security Attack in the Continuous-Time World

Shuling Wang[1], Flemming Nielson[2], and Hanne Riis Nielson[2]

[1] State Key Laboratory of Computer Science, Institute of Software
Chinese Academy of Sciences, China
[2] DTU Informatics, Technical University of Denmark, Denmark

**Abstract.** Hybrid systems are integrations of discrete computation and continuous physical evolution. The physical components of such systems introduce safety requirements, the achievement of which asks for the correct monitoring and control from the discrete controllers. However, due to denial-of-service security attack, the expected information from the controllers is not received and as a consequence the physical systems may fail to behave as expected. This paper proposes a formal framework for expressing denial-of-service security attack in hybrid systems. As a virtue, a physical system is able to plan for reasonable behavior in case the ideal control fails due to unreliable communication, in such a way that the safety of the system upon denial-of-service is still guaranteed. In the context of the modeling language, we develop an inference system for verifying safety of hybrid systems, without putting any assumptions on how the environments behave. Based on the inference system, we implement an interactive theorem prover and have applied it to check an example taken from train control system.

**Keywords:** Hybrid systems, Denial-of-service, Safety verification, Inference system

## 1 Introduction

Hybrid systems, also known as cyber-physical systems, are dynamic systems with interacting continuous-time physical systems and discrete controllers. The physical systems evolve continuously with respect to time, such as aircrafts, or biological cell growth, while the computer controllers, such as autopilots, or biological control circuits, monitor and control the behavior of the systems to meet the given design requirements. One design requirement is safety, which includes time-critical constraints, or invariants etc., for the example of train control systems, the train should arrive at the stops on time, or the train must always move with a velocity within a safe range.

However, due to the uncertainty in the environment, the potential errors in wireless communications between the interacting components will make the safety of the system very hard to guarantee. For the sake of safety, when the controllers fail to behave as expected because of absence of expected communication and thus become unreliable, the physical systems should provide feedback control, to achieve safety despite errors in communication.

***A Motivating Example*** We illustrate our motivation by an example taken from train control system, that is depicted in Fig. 1. It consists of three inter-communicating components: Train, Driver and on board vital computer (VC). We assume that the train owns arbitrarily long movement authority, within which the train is allowed to move only, and must conform to a safety requirement, i.e. the velocity must be non-negative and cannot exceed a maximum limit. The train acts as a continuous plant, and moves with a given acceleration; both the driver and the VC act as controllers, in such a way that, either of them observes the velocity of the train periodically, and then according to the safety requirement, computes the new acceleration for the train to follow in the next period. According to the specification of the system, the message from the VC always takes high priority over the one from the driver.

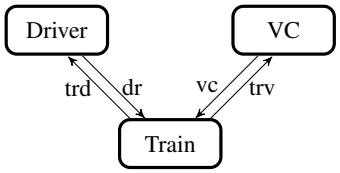


**Fig. 1.** The structure of train control example

However, the expected monitoring and control from VC or driver may fail due to denial-of-service security attack, e.g. if the driver falls asleep, or if the VC gets malfunction, and as a consequence, the train may get no response from any of them within a duration of time. The safety requirement of the train will then be violated very easily. This poses the problem of how to build a safe hybrid system in the presence of this sort of denial-of-service security attack from the environment.

The contribution of this paper includes the following aspects:

- a programming notation, for formally modeling hybrid systems and meanwhile being able to express denial-of-service due to unreliable communications, and an assertion language, for describing safety as annotations in such programs;
- a deductive inference system, for reasoning about whether the program satisfies the annotated safety property, and a subsequent interactive theorem prover.

As a direct application, we are able to build a safe system for the example such that:

(F1) the error configurations where neither driver nor VC is available are not reachable;
(F2) the velocity of the train keeps always in the safe range, although in the presence of denial-of-service attack from the driver or the VC.

Furthermore, when the behavior of the environments (i.e. driver and VC) is determined, e.g. by defining some constraints among the constants of the whole system, we can learn more precise behavior of the train.

In Section 2 and Section 3, we present the syntax and semantics for the formal modeling language. It is a combination of Hybrid CSP (HCSP) [5, 20], a process algebra based modeling language for describing hybrid systems, and the binders from Quality Calculus [13], a process calculus that allows one to take measures in case of unreliable

communications. With the introducing of binders, the modelling language is capable of programming a safe system that executes in an open environment that does not always live up to expectations.

In Section 4, we define an inference system for reasoning about HCSP with binders. For each construct $P$, the specification is of the form $\{\varphi\}\ P\ \{\psi, HF\}$, where $\varphi$ and $\psi$ are the pre-/post-condition recording the initial and terminating states of $P$ respectively, and $HF$ the history formula recording the whole execution history of $P$ (thus able to specify global invariants). As a direct application, the (un-)reachability analysis can be performed by identifying the points corresponding to the error configurations by logical formulas and then checking the (un-)satisfiability of the formulas. In Section 5, we have applied a theorem prover we have implemented to verify properties (F1) and (F2) of the train control example. At last, we conclude the paper and address some future work.

***Related Work*** There have been numerous work on formal modeling and verification of hybrid systems, e.g., [1, 11, 6, 10, 7], the most popular of which is *hybrid automata* [1, 11, 6]. For automata-based approaches, the verification of hybrid systems is reduced to computing reachable sets, which is conducted either by model-checking [1] or by the decision procedure of Tarski algebra [7]. However, hybrid automata, analogous to state machines, has little support for structured description; and meanwhile, the verification of it based on reachability computation is not scalable and only applicable to some specific linear hybrid systems, as it heavily depends on the decidability of the problems to be solved. Applying abstraction or (numeric) approximation [4, 2, 3] can improve the scalability, but as a pay we have to sacrifice the precision.

In contrast, deductive methods increasingly attract more attention in the verification of hybrid systems as it can scale up to complex systems. A differential-algebraic dynamic logic for hybrid programs [14] was proposed by extending dynamic logic with continuous statements, and has been applied for safety checking of European Train Control System [15]. However, the hybrid programs there can be considered as a textual encoding of hybrid automata, with no support for communication and concurrency. In [8, 17], the Hoare logic is extended to hybrid systems modeled by Hybrid CSP [5, 20], and then used for safety checking of Chinese Train Control System. But the logic lacks compositionality.

All the work mentioned above focus on safety without considering denial-of-service security attacks from the environment. Quality Calculus [13, 12] for the first time proposed a programming notation for expressing denial-of-service in communication systems, but is currently limited to discrete time world.

## 2 Syntax

We first choose Hybrid CSP (HCSP) [5, 20] as the modelling language for hybrid systems. HCSP inherits from CSP the explicit communication model and concurrency, thus is expressive enough for describing distributed components and the interactions between them. Moreover, it extends CSP with differential equations for representing continuous evolution, and provides several forms of interrupts to continuous evolution for realizing communication-based discrete control. On the other hand, Quality

Calculus [13, 12] is recently proposed to programming software components and their interactions in the presence of unreliable communications. With the help of *binders* specifying the success or failure of communications and then the communications to be performed before continuing, it becomes natural in Quality Calculus to plan for reasonable behavior in case the ideal behavior fails due to unreliable communication and thereby to increase the quality of the system.

In our approach, we will extend HCSP further with the notion of binders from Quality Calculus, for modelling hybrid systems in the presence of denial-of-service because of unreliable communications. The overall modelling language is given by the following syntax:

$$
\begin{array}{lll}
e & ::= & c \mid x \mid f^k(e_1, ..., e_k) \\
b & ::= & ch!e\{u_1\} \mid ch?x\{u_2\} \mid \&_q(b_1, \cdots, b_n) \\
P, Q & ::= & \text{skip} \mid x := e \mid b \mid \langle \mathcal{F}(\dot{s}, s) = 0 \& B \rangle \mid \langle \mathcal{F}(\dot{s}, s) = 0 \& B \rangle \trianglerighteq b \to Q \mid \\
& & P \| Q \mid P; Q \mid \omega \to P \mid P^*
\end{array}
$$

Expressions $e$ are used to construct data elements and consist of constants $c$, data variables $x$, and function application $f^k(e_1, ..., e_k)$.

Binders $b$ specify the inputs and outputs to be performed before continuing. The output $ch!e\{u_1\}$ expects to send message $e$ along channel $ch$, with $u_1$ being the acknowledgement in case the communication succeeds, and the dual input $ch?x\{u_2\}$ expects to receive a message from $ch$ and assigns it to variable $x$, with $u_2$ being the acknowledgement similarly. We call both $u_1$ and $u_2$ *acknowledgment variables*, and assume in syntax that for each input or output statement, there exists a unique acknowledgement variable attached to it. In the sequel, we will use $\mathcal{V}$ and $\mathcal{A}$ to represent the set of data variables and acknowledgement variables respectively, and they are disjoint. For the general form $\&_q(b_1, \cdots, b_n)$, the quality predicate $q$ specifies the sufficient communications among $b_1, \cdots, b_n$ for the following process to proceed. In syntax, $q$ is a logical combination of quality predicates corresponding to $b_1, \cdots, b_n$ recursively (denoted by $q_1, \cdots, q_n$ respectively below). For example, the quality predicates for $ch!e\{u_1\}$ and $ch?x\{u_2\}$ are boolean formulas $u_1 = 1$ and $u_2 = 1$. There are two special forms of quality predicates, abbreviated as $\exists$ and $\forall$, with the definitions: $\forall \stackrel{\text{def}}{=} q_1 \wedge \cdots \wedge q_n$ and $\exists \stackrel{\text{def}}{=} q_1 \vee \cdots \vee q_n$. More forms of quality predicates can be found in [13].

*Example 1.* For the train example, define binder $b_0$ as $\&_\exists(\mathsf{dr}?x_a\{u_a\}, \mathsf{vc}?y_a\{w_a\})$, the quality predicate of which amounts to $u_a = 1 \vee w_a = 1$. It expresses that, the train is waiting for the acceleration from the driver and the VC, via dr and vc respectively, and as soon as one of the communications succeeds (i.e., when the quality predicate becomes true), the following process will be continued without waiting for the other. □

$P, Q$ define processes. The skip and assignment $x := e$ are defined as usual, taking no time to complete. Binders $b$ are explained above. The continuous evolution $\langle \mathcal{F}(\dot{s}, s) = 0 \& B \rangle$, where $s$ represents a vector of continuous variables and $\dot{s}$ the corresponding first-order derivative of $s$, forces $s$ to evolve according to the differential equations $\mathcal{F}$ as long as $B$, a boolean formula of $s$ that defines the *domain of* $s$, holds, and terminates when $B$ turns false. The communication interrupt $\langle \mathcal{F}(\dot{s}, s) = 0 \& B \rangle \trianglerighteq b \to Q$ behaves as $\langle \mathcal{F}(\dot{s}, s) = 0 \& B \rangle$ first, and if $b$ occurs before the continuous terminates, the continuous will be preempted and $Q$ will be executed instead.

The rest of the constructs define compound processes. The parallel composition $P \| Q$ behaves as if $P$ and $Q$ run independently except that the communications along the common channels connecting $P$ and $Q$ are to be synchronized. In syntax, $P$ and $Q$ in parallel are restricted not to share variables, nor input or output channels. The sequential composition $P; Q$ behaves as $P$ first, and if it terminates, as $Q$ afterwards. The conditional $\omega \rightarrow P$ behaves as $P$ if $\omega$ is true, otherwise terminates immediately. The condition $\omega$ can be used for checking the status of data variables or acknowledgement, thus in syntax, it is a boolean formula on data and acknowledgement variables (while for the above continuous evolution, $B$ is a boolean formula on only data variables). The repetition $P^*$ executes $P$ for arbitrarily finite number of times.

It should be noticed that, with the addition of binders, it is able to derive a number of other known constructs of process calculi, e.g., internal and external choice [13].

*Example 2.* Following Example 1, the following model

$$t := 0; {}^1\langle \dot{s} = v, \dot{v} = a, \dot{t} = 1 \& t < T \rangle \trianglerighteq {b_0}^2 \rightarrow \\ (w_a = 1^3 \rightarrow a := y_a; w_a = 0 \wedge u_a = 1^4 \rightarrow a := x_a;\ \ w_a = 0 \wedge u_a = 0^5 \rightarrow \text{skip})$$

denoted by $P_0$, expresses that, the train moves with velocity $v$ and acceleration $a$, and as soon as $b_0$ occurs within $T$ time units, i.e. the train succeeds to receive a new acceleration from either the driver or the VC, then its acceleration $a$ will be updated by case analysis. It can be seen that the acceleration from VC will be used in priority. For later reference we have annotated the program with labels (e.g. 1, 2, etc.). □

## 3 Transition Semantics

We first introduce a variable $now$ to record the global time during process execution, and then define the set $\mathcal{V}^+ = \mathcal{V} \cup \mathcal{A} \cup \{now\}$. A state, ranging over $\sigma, \sigma'$, assigns a value to each variable in $\mathcal{V}^+$, and we will use $\Sigma$ to represent the set of states. A flow, ranging over $h, h'$, defined on a closed time interval $[r_1, r_2]$ with $0 \leq r_1 \leq r_2$, or an infinite interval $[r, \infty)$ with some $r \geq 0$, assigns a state in $\Sigma$ to each point in the interval. Given a state $\sigma$, an expression $e$ is evaluated to a value under $\sigma$, denoted by $\sigma(e)$ below.

Each transition relation has the form $(P, \sigma) \xrightarrow{\alpha} (P', \sigma', h)$, where $P$ is a process, $\sigma, \sigma'$ are states, $h$ is a flow, and $\alpha$ is an event. It represents that starting from initial state $\sigma$, $P$ evolves into $P'$ and ends with state $\sigma'$ and flow $h$, while performing event $\alpha$. When the above transition takes no time, it produces a point flow, i.e. $\sigma(now) = \sigma'(now)$ and $h = \{\sigma(now) \mapsto \sigma'\}$, and we will call the transition *discrete* and write $(P, \sigma) \xrightarrow{\alpha} (P', \sigma')$ instead without losing any information. The label $\alpha$ represents events, which can be a discrete internal event, like skip, assignment, evaluation of boolean conditions, or termination of a continuous evolution etc., uniformly denoted by $\tau$, or an external communication, like output $ch!c\{1\}$ or input $ch?c\{1\}$, or an internal communication $ch\dagger c\{1\}$, or a time delay $d$ for some positive $d$. We call the events but the time delay *discrete events*, and will use $\beta$ to range over them.

The transition relations for binders are defined in Table 1. The input $ch?x\{u\}$ may perform an external communication $ch?c\{1\}$, and as a result $x$ will be bound to $c$ and $u$ set to 1, or it may keep waiting for $d$ time. For the second case, a flow

$$(ch?x\{u\},\sigma) \xrightarrow{ch?c\{1\}} (\epsilon, \sigma[x \mapsto c, u \mapsto 1])$$

$$(ch?x\{u\},\sigma) \xrightarrow{d} (ch?x\{u\}, \sigma[now + d], h_d)$$

$$(ch!e\{u\},\sigma) \xrightarrow{ch!\sigma(e)\{1\}} (\epsilon, \sigma[u \mapsto 1]) \quad (ch!e\{u\},\sigma) \xrightarrow{d} (ch!e\{u\}, \sigma[now + d], h_d)$$

---

$$[\![q]\!](b_1, \cdots, b_n) = q[(b_1 \equiv \epsilon)/q_1, \cdots, (b_n \equiv \epsilon)/q_n]$$

$$\langle\!|()|\!\rangle\sigma = \sigma \quad \langle\!|(\epsilon, b_2, \cdots, b_n)|\!\rangle\sigma = \langle\!|(b_2, \cdots, b_n)|\!\rangle\sigma$$

$$\langle\!|(ch?x\{u\}, b_2, \cdots, b_n)|\!\rangle\sigma = \langle\!|(b_2, \cdots, b_n)|\!\rangle(\sigma[u \mapsto 0])$$

$$\langle\!|(ch!e\{u\}, b_2, \cdots, b_n)|\!\rangle\sigma = \langle\!|(b_2, \cdots, b_n)|\!\rangle(\sigma[u \mapsto 0])$$

$$\langle\!|(\&_{q_k}(b_{k1}, \cdots, b_{km}), b_2, \cdots, b_n)|\!\rangle\sigma = \langle\!|(b_{k1}, \cdots, b_{km}, b_2, \cdots, b_n)|\!\rangle\sigma$$

---

$$\frac{[\![q]\!](b_1, \cdots, b_n) = false}{(\&_q(b_1, \cdots, b_n), \sigma) \xrightarrow{d} (\&_q(b_1, \cdots, b_n), \sigma[now + d], h_d)}$$

$$\frac{(b_i, \sigma) \xrightarrow{\beta} (b_i', \sigma')}{(\&_q(b_1, \cdots, b_i, \cdots, b_n), \sigma) \xrightarrow{\beta} (\&_q(b_1, \cdots, b_i', \cdots, b_n), \sigma')}$$

$$\frac{[\![q]\!](b_1, \cdots, b_n) = true \quad \langle\!|(b_1, \cdots, b_n)|\!\rangle\sigma = \sigma'}{(\&_q(b_1, \cdots, b_n), \sigma) \xrightarrow{\tau} (\epsilon, \sigma')}$$

---

**Table 1.** The transition relations for binders and the auxiliary functions

$h_d$ over $[\sigma(now), \sigma(now) + d]$ is produced, satisfying that for any $t$ in the domain, $h_d(t) = \sigma[now \mapsto t]$, i.e. no variable but the clock $now$ in $\mathcal{V}^+$ is changed during the waiting period. Similarly, there are two rules for output $ch!e\{u\}$. Here $\sigma[now + d]$ is an abbreviation for $\sigma[now \mapsto \sigma(now) + d]$.

Before defining the semantics of general binders, we introduce two auxiliary functions. Assume $(b_1, \cdots, b_n)$ is an intermediate tuple of binders that occurs during execution (thus some of $b_i$s might contain $\epsilon$), $q$ a quality predicate, and $\sigma$ a state. The function $[\![q]\!](b_1, \cdots, b_n)$ defines the truth value of $q$ under $(b_1, \cdots, b_n)$, which is calculated by replacing each sub-predicate $q_i$ corresponding to $b_i$ in $q$ by $b_i \equiv \epsilon$ respectively; and function $\langle\!|(b_1, b_2, \cdots, b_n)|\!\rangle\sigma$ returns a state that fully reflects the failure or success of binders $b_1, \cdots, b_n$, and can be constructed from $\sigma$ by setting the acknowledgement variables corresponding to the failing inputs or outputs among $b_1, \cdots, b_n$ to be 0. Based on these definitions, binder $\&_q(b_1, \cdots, b_n)$ may keep waiting for $d$ time, if $q$ is false under $(b_1, \cdots, b_n)$, or perform a discrete event $\beta$ that is enabled for some $b_i$, or perform a $\tau$ transition and terminate if $q$ is true under $(b_1, \cdots, b_n)$. Notice that when $q$ becomes true, the enabled discrete events can still be performed, as indicated by the second rule.

*Example 3.* Starting from $\sigma_0$, the execution of $b_0$ in Example 1 may lead to three possible states at termination:

- $\sigma_0[now + d, x_a \mapsto c_a, u_a \mapsto 1, w_a \mapsto 0]$, indicating that the train succeeds to receive $c_a$ from the driver after $d$ time units have passed, but fails for the VC;
- $\sigma_0[now + d, y_a \mapsto d_a, w_a \mapsto 1, u_a \mapsto 0]$, for the opposite case of the first;
- $\sigma_0[now + d, x_a \mapsto c_a, u_a \mapsto 1, y_a \mapsto d_a, w_a \mapsto 1]$, indicating that the train succeeds to receive messages from the driver as well as the VC after $d$ time. □

The transition relations for other processes are defined in Table 2. The rules for skip and assignment can be defined as usual. The idle rule represents that the process can stay at the terminating state $\epsilon$ for arbitrary $d$ time units, with nothing changed but only the clock progress. For continuous evolution, for any $d > 0$, it evolves for $d$ time units according to $\mathcal{F}$ if $B$ evaluates to true within this period (the right end exclusive). A flow $h_{d,s}$ over $[\sigma(now), \sigma(now) + d]$ will then be produced, such that for any $o$ in the domain, $h_{d,s}(o) = \sigma[now \mapsto o, s \mapsto S(o - \sigma(now))]$, where $S(t)$ is the solution as defined in the rule. Otherwise, the continuous evolution terminates at a point if $B$ evaluates to false at the point, or if $B$ evaluates to false at a positive open interval right to the point.

For communication interrupt, the process may evolve for $d$ time units if both the continuous evolution and the binder can progress for $d$ time units, and then reach the same state and flow as the continuous evolution does. It may perform a discrete event over $b$, and if the resulting binder $b'$ is not $\epsilon$, then the continuous evolution is kept, otherwise, the continuous evolution will be interrupted and $Q$ will be followed to execute, and for both cases, will reach the same state and flow as the binder does. Finally, it may perform a $\tau$ event and terminate immediately if the continuous evolution terminates with a $\tau$ event but $b$ not. Notice that the final state $\sigma''$ needs to be reconstructed from $\sigma'$ by resetting the acknowledgement variables of those unsuccessful binders occurring in $b$ to be 0.

Before defining the semantics of parallel composition, we need to introduce some notations. Two states $\sigma_1$ and $\sigma_2$ are *disjoint*, iff $\text{dom}(\sigma_1) \cap \text{dom}(\sigma_2) = \{now\}$ and $\sigma_1(now) = \sigma_2(now)$. For two disjoint states $\sigma_1$ and $\sigma_2$, $\sigma_1 \uplus \sigma_2$ is defined as a state over $\text{dom}(\sigma_1) \cup \text{dom}(\sigma_2)$, satisfying that $\sigma_1 \uplus \sigma_2(v)$ is $\sigma_1(v)$ if $v \in \text{dom}(\sigma_1)$, otherwise $\sigma_2(v)$ if $v \in \text{dom}(\sigma_2)$. We lift this definition to flows $h_1$ and $h_2$ satisfying $\text{dom}(h_1) = \text{dom}(h_2)$, and define $h_1 \uplus h_2$ to be a flow such that $h_1 \uplus h_2(t) = h_1(t) \uplus h_2(t)$. For $P \| Q$, assume $\sigma_1$ and $\sigma_2$ represent the initial states for $P$ and $Q$ respectively and are disjoint. The process will perform a communication along a common channel of $P$ and $Q$, if $P$ and $Q$ get ready to synchronize with each other along the channel. Otherwise, it will perform a discrete event, that can be $\tau$, an internal communication of $P$, or an external communication along some non-common channel of $P$ and $Q$, if $P$ can progress separately on this event (and the symmetric rule for $Q$ is left out here). When neither internal communication nor $\tau$ event is enabled for $P||Q$, it may evolve for $d$ time units if both $P$ and $Q$ can evolve for $d$ time units. Finally, the process will perform a $\tau$ event and terminate as soon as both the components terminate.

At last, the rules for conditional, sequential, and repetition are defined as usual.

*Example 4.* Starting from state $\sigma_0$, the execution of $P_0$ in Example 2 leads to the following cases (let $v_0$ denote $\sigma_0(v)$ below):

- $P_0$ terminates without the occurrence of $b_0$, the final state is $\sigma_0[now+T, t+T, v+aT, s + v_0T + 0.5aT^2, u_a \mapsto 0, w_a \mapsto 0]$;

**Skip, Assignment and Idle** $(\text{skip}, \sigma) \xrightarrow{\tau} (\epsilon, \sigma)$

$$(x := e, \sigma) \xrightarrow{\tau} (\epsilon, \sigma[x \mapsto \sigma(e)]) \quad (\epsilon, \sigma) \xrightarrow{d} (\epsilon, \sigma[now + d], h_d)$$

---

**Continuous Evolution** For any $d > 0$,

$$\frac{\begin{array}{c}S(t) \text{ is a solution of } \mathcal{F}(\dot{s}, s) = 0 \text{ over } [0, d] \text{ satisfying that } S(0) = \sigma(s) \\ \text{and } \forall t \in [0, d).h_{d,s}(t + \sigma(now))(B) = true\end{array}}{(\langle \mathcal{F}(\dot{s}, s) = 0 \& B\rangle, \sigma) \xrightarrow{d} (\langle \mathcal{F}(\dot{s}, s) = 0 \& B\rangle, \sigma[now + d, s \mapsto S(d)], h_{d,s})}$$

$$\frac{\begin{array}{c}(\sigma(B) = false) \text{ or } (\sigma(B) = true \wedge \exists \delta > 0. \\ (S(t) \text{ is a solution of } \mathcal{F}(\dot{s}, s) = 0 \text{ over } [0, \delta] \text{ satisfying that } S(0) = \sigma(s) \\ \text{and } \forall t \in (0, \delta).h_{\delta,s}(t + \sigma(now))(B) = false))\end{array}}{(\langle \mathcal{F}(\dot{s}, s) = 0 \& B\rangle, \sigma) \xrightarrow{\tau} (\epsilon, \sigma)}$$

---

**Communication Interrupt**

$$\frac{(\langle \mathcal{F}(\dot{s}, s) = 0 \& B\rangle, \sigma) \xrightarrow{d} (\langle \mathcal{F}(\dot{s}, s) = 0 \& B\rangle, \sigma', h) \quad (b, \sigma) \xrightarrow{d} (b, \sigma'', h'')}{(\langle \mathcal{F}(\dot{s}, s) = 0 \& B\rangle \trianglerighteq b \to Q, \sigma) \xrightarrow{d} (\langle \mathcal{F}(\dot{s}, s) = 0 \& B\rangle \trianglerighteq b \to Q, \sigma', h)}$$

$$\frac{(b, \sigma) \xrightarrow{\beta} (b', \sigma') \quad b' \neq \epsilon}{(\langle \mathcal{F}(\dot{s}, s) = 0 \& B\rangle \trianglerighteq b \to Q, \sigma) \xrightarrow{\beta} (\langle \mathcal{F}(\dot{s}, s) = 0 \& B\rangle \trianglerighteq b' \to Q, \sigma')}$$

$$\frac{(b, \sigma) \xrightarrow{\beta} (\epsilon, \sigma')}{(\langle \mathcal{F}(\dot{s}, s) = 0 \& B\rangle \trianglerighteq b \to Q, \sigma) \xrightarrow{\beta} (Q, \sigma')}$$

$$\frac{\begin{array}{c}(\langle \mathcal{F}(\dot{s}, s) = 0 \& B\rangle, \sigma) \xrightarrow{\tau} (\epsilon, \sigma') \quad \neg((b, \sigma) \xrightarrow{\tau} (\epsilon, -)) \\ b \equiv \&_q(b_1, \cdots, b_n) \quad \langle\!|(b_1, \cdots, b_n)|\!\rangle \sigma' = \sigma''\end{array}}{(\langle \mathcal{F}(\dot{s}, s) = 0 \& B\rangle \trianglerighteq b \to Q, \sigma) \xrightarrow{\tau} (\epsilon, \sigma'')}$$

---

**Parallel Composition**

$$\frac{(P, \sigma_1) \xrightarrow{ch?c\{1\}} (P', \sigma_1') \quad (Q, \sigma_2) \xrightarrow{ch!c\{1\}} (Q', \sigma_2')}{(P \parallel Q, \sigma_1 \uplus \sigma_2) \xrightarrow{ch\dagger c\{1\}} (P' || Q', \sigma_1' \uplus \sigma_2')}$$

$$\frac{\begin{array}{c}(P, \sigma_1) \xrightarrow{\beta} (P', \sigma_1') \quad \beta \in \{\tau, ch\dagger c\{1\}, ch?c\{1\}, ch!c\{1\} \mid \\ ch \notin \mathbf{Chan}(P) \cap \mathbf{Chan}(Q)\} \quad \forall ch, c.(\neg((P, \sigma_1) \xrightarrow{ch?c\{1\}} \wedge (Q, \sigma_2) \xrightarrow{ch!c\{1\}}) \\ \wedge \neg((P, \sigma_1) \xrightarrow{ch!c\{1\}} \wedge (Q, \sigma_2) \xrightarrow{ch?c\{1\}}))\end{array}}{(P \parallel Q, \sigma_1 \uplus \sigma_2) \xrightarrow{\beta} (P' || Q, \sigma_1' \uplus \sigma_2)}$$

$$\frac{\begin{array}{c}(P, \sigma_1) \xrightarrow{d} (P', \sigma_1', h_1) \quad (Q, \sigma_2) \xrightarrow{d} (Q', \sigma_2', h_2) \\ \forall ch, c.\neg((P \parallel Q, \sigma_1 \uplus \sigma_2) \xrightarrow{ch\dagger c\{1\}}) \quad \neg((P \parallel Q, \sigma_1 \uplus \sigma_2) \xrightarrow{\tau})\end{array}}{(P \parallel Q, \sigma_1 \uplus \sigma_2) \xrightarrow{d} (P' || Q', \sigma_1' \uplus \sigma_2', h_1 \uplus h_2)}$$

$$(\epsilon \| \epsilon, \sigma) \xrightarrow{\tau} (\epsilon, \sigma)$$

---

**Other Compound Constructs**

$$\frac{\sigma(\omega) = true}{(\omega \to P, \sigma) \xrightarrow{\tau} (P, \sigma)} \quad \frac{\sigma(\omega) = false}{(\omega \to P, \sigma) \xrightarrow{\tau} (\epsilon, \sigma)}$$

$$\frac{(P, \sigma) \xrightarrow{\alpha} (P', \sigma', h) \quad P' \neq \epsilon}{(P; Q, \sigma) \xrightarrow{\alpha} (P'; Q, \sigma', h)} \quad \frac{(P, \sigma) \xrightarrow{\alpha} (\epsilon, \sigma', h)}{(P; Q, \sigma) \xrightarrow{\alpha} (Q, \sigma', h)}$$

$$\frac{(P, \sigma) \xrightarrow{\alpha} (P', \sigma', h) \quad P' \neq \epsilon}{(P^*, \sigma) \xrightarrow{\alpha} (P'; P^*, \sigma', h)} \quad \frac{(P, \sigma) \xrightarrow{\alpha} (\epsilon, \sigma', h)}{(P^*, \sigma) \xrightarrow{\alpha} (P^*, \sigma', h)} \quad (P^*, \sigma) \xrightarrow{\tau} (\epsilon, \sigma)$$

**Table 2.** The transition relations for processes

– $b_0$ occurs after $d$ time units for some $d \leq T$, and as a result $P_0$ executes to location 2, with state $\sigma_0[now + d, t + d, v + ad, s + v_0 d + 0.5ad^2, u_a, w_a, x_a, y_a]$, where $u_a$, $w_a$, $x_a$ and $y_a$ have 3 possible evaluations as defined in Example 3, and then depending on the values of $u_a$ and $w_a$, executes to location 3 or 4 respectively, and finally terminates after a corresponding acceleration update. □

***Flow of a Process*** Given two flows $h_1$ and $h_2$ defined on $[r_1, r_2]$ and $[r_2, r_3]$ (or $[r_2, \infty)$) respectively, we define the concatenation $h_1^\frown h_2$ as the flow defined on $[r_1, r_3]$ (or $[r_1, \infty)$) such that $h_1^\frown h_2(t)$ is equal to $h_1(t)$ if $t \in [r_1, r_2)$, otherwise $h_2(t)$. Given a process $P$ and an initial state $\sigma$, if we have the following sequence of transitions:

$$\begin{array}{l}(P, \sigma) \xrightarrow{\alpha_0} (P_1, \sigma_1, h_1) \quad (P_1, \sigma_1) \xrightarrow{\alpha_1} (P_2, \sigma_2, h_2) \\ \ldots \quad (P_{n-1}, \sigma_{n-1}) \xrightarrow{\alpha_{n-1}} (P_n, \sigma_n, h_n)\end{array}$$

then we define $h_1^\frown \ldots^\frown h_n$ as the *flow* from $P$ to $P_n$ with respect to the initial state $\sigma$, and furthermore, write $(P, \sigma) \xrightarrow{\alpha_0 \cdots \alpha_{n-1}} (P_n, \sigma_n, h_1^\frown \ldots^\frown h_n)$ to represent the whole transition sequence (and for simplicity, the label sequence can be omitted sometimes). When $P_n$ is $\epsilon$, we call $h_1^\frown \ldots^\frown h_n$ a *complete flow* of $P$ with respect to $\sigma$.

## 4 Inference System

In this section, we define an inference system for reasoning about both discrete and continuous properties of HCSP with binders, which are considered for an isolated time point and a time interval respectively.

***History Formulas*** In order to describe the interval-related properties, we introduce history formulas, that are defined by duration calculus (DC) [19, 18]. DC is a first-order interval-based real-time logic with one binary modality known as chop $\frown$. History formulas $HF$ are defined by the following subset of DC:

$$HF ::= \ell \circ T \mid \lceil S \rceil \mid HF_1 \frown HF_2 \mid \neg HF \mid HF_1 \vee HF_2$$

where $\ell$ is a temporal variable denoting the length of the considered interval, $\circ \in \{<, =\}$ is a relation, $T$ a non-negative real, and $S$ a first-order state formula over process variables. For simplicity, we will write $\lceil S \rceil^<$ as an abbreviation for $\lceil S \rceil \vee \ell = 0$.

$HF$ can be interpreted over flows and intervals. We define the judgement $h, [a, b] \models HF$ to represent that $HF$ holds under $h$ and $[a, b]$, then we have

$$\begin{array}{l}h, [a, b] \models \ell \circ T \text{ iff } (b - a) \circ T \qquad h, [a, b] \models \lceil S \rceil \text{ iff } \int_a^b h(t)(S) = b - a \\ h, [a, b] \models HF_1 \frown HF_2 \text{ iff } \exists c. a \leq c \leq b \wedge h, [a, c] \models HF_1 \wedge h, [c, b] \models HF_2\end{array}$$

As defined above, $\ell$ indicates the length of the considered interval; $\lceil S \rceil$ asserts that $S$ holds almost everywhere in the considered interval; and $HF_1 \frown HF_2$ asserts that the interval can be divided into two sub-intervals such that $HF_1$ holds for the first and $HF_2$ for the second. The first-order connectives $\neg$ and $\vee$ can be explained as usual.

All axioms and inference rules for DC presented in [18] can be applied here, such as

$$\text{True} \Leftrightarrow \ell \geq 0 \quad \lceil S \rceil^\frown \lceil S \rceil \Leftrightarrow \lceil S \rceil \quad HF^\frown \ell = 0 \Leftrightarrow HF$$
$$\lceil S_1 \rceil \Rightarrow \lceil S_2 \rceil \text{ if } S_1 \Rightarrow S_2 \text{ is valid in FOL}$$

***Specification*** The specification for process $P$ takes form $\{\varphi\}\ P\ \{\psi, HF\}$, where the pre-/post-condition $\varphi$ and $\psi$, defined by FOL, specify properties of variables that hold at the beginning and termination of the execution of $P$ respectively, and the history formula $HF$, specifies properties of variables that hold throughout the execution interval of $P$. The specification of $P$ is defined with no dependence on the behavior of its environment. The specification is *valid*, denoted by $\models \{\varphi\}\ P\ \{\psi, HF\}$, iff for any state $\sigma$, if $(P, \sigma) \rightarrow (\epsilon, \sigma', h)$, then $\sigma \models \varphi$ implies $\sigma' \models \psi$ and $h, [\sigma(now), \sigma'(now)] \models HF$.

***Acknowledgement of Binders*** In order to define the inference rules for binders $b$, we first define an auxiliary typing judgement $\vdash b \blacktriangleright \varphi$, where the first-order formula $\varphi$ describes the acknowledgement corresponding to successful passing of $b$, and is defined without dependence on the precondition of $b$. We say $b \blacktriangleright \varphi$ *valid*, denoted by $\models b \blacktriangleright \varphi$, iff given any state $\sigma$, if $(b, \sigma) \rightarrow (\epsilon, \sigma', h)$, then $\sigma' \models \varphi$ holds.

The typing judgement for binders is defined as follows:

$$\vdash ch?x\{u\} \blacktriangleright u = 1 \quad \vdash ch!e\{u\} \blacktriangleright u = 1 \quad \frac{\vdash b_1 \blacktriangleright \varphi_1, \cdots, \vdash b_n \blacktriangleright \varphi_n}{\vdash \&_q(b_1, \cdots, b_n) \blacktriangleright [\![q]\!](\varphi_1, \cdots, \varphi_n)}$$

As indicated above, for input $ch?x\{u\}$, the successful passing of it gives rise to formula $u = 1$, and similarly for output $ch!e\{u\}$; for binder $\&_q(b_1, \cdots, b_n)$, it gives rise to formula $[\![q]\!](\varphi_1, \cdots, \varphi_n)$, which encodes the effect of quality predicate $q$ to the subformulas $\varphi_1, \ldots, \varphi_n$ corresponding to $b_1, \ldots, b_n$ respectively.

*Example 5.* For binder $b_0$ in Example 1, we have $\vdash b_0 \blacktriangleright u_a = 1 \vee w_a = 1$, indicating that, if the location after $b_0$ is reachable, then at least one of the communications with the driver or the VC succeeds. □

### 4.1 Inference Rules

We first introduce an auxiliary function $mv(b)$, which given a binder $b$, returns the variables that may be modified by $b$. It can be defined directly by structural induction on $b$ and we omit the details. The inference rules for deducing the specifications of all constructs are presented in Table 3.

Statements skip and assignment are defined as in classical Hoare Logic, plus $\ell = 0$ in the history formula, indicating that they both take zero time to complete. For each form of the binders $b$, the postcondition is the conjunction of the quantified precondition $\varphi$ over variables in $mv(b)$ and the acknowledgement corresponding to the successful passing of $b$. The binders may occur without waiting any time, indicated by $\ell = 0$ as one disjunctive clause of each history formula. For both $ch?x\{u\}$ and $ch!e\{u\}$, if the waiting time is greater than 0, then $\varphi$ will hold almost everywhere in the waiting interval (the only possible exception is the right endpoint, at which the communication occurs

$$\{\varphi\}\ \text{skip}\ \{\varphi, \ell = 0\} \quad \{\psi[e/x]\}\ x := e\ \{\psi, \ell = 0\}$$

$$\{\varphi\}\ ch?x\{u\}\ \{(\exists x, u.\varphi) \wedge u = 1, \lceil\varphi\rceil^{<}\}\ \ \{\varphi\}\ ch!e\{u\}\ \{(\exists u.\varphi) \wedge u = 1, \lceil\varphi\rceil^{<}\}$$

$$\frac{\vdash \&_q(b_1, \cdots, b_n) \blacktriangleright \alpha}{\{\varphi\}\ \&_q(b_1, \cdots, b_n)\ \{\ (\exists mv(\&_q(b_1, \cdots, b_n)).\varphi) \wedge \alpha, \lceil\exists mv(\&_q(b_1, \cdots, b_n)).\varphi\rceil^{<}\ \}}$$

$$\{\varphi\}\ \langle \mathcal{F}(\dot{s}, s) = 0 \& B\rangle\ \{\ (\exists s.\varphi) \wedge cl(\neg B) \wedge cl(Inv), \lceil(\exists s.\varphi) \wedge B \wedge Inv\rceil^{<}\ \}$$

$$\frac{\vdash \&_q(b_1, \cdots, b_n) \blacktriangleright \alpha \quad \{(\exists mv(b).(\exists s.\varphi) \wedge cl(Inv)) \wedge \alpha\}\ Q\ \{\psi_1, h_1\}}{\{\varphi\}\langle \mathcal{F}(\dot{s}, s) = 0 \& B\rangle \trianglerighteq b \rightarrow Q\ \begin{array}{l}\{(\exists mv(b).(\exists s.\varphi) \wedge cl(\neg B) \wedge cl(Inv)) \vee \psi_1,\\ \lceil\exists mv(b).(\exists s.\varphi) \wedge B \wedge Inv\rceil^{<} \widehat{\ } (\ell = 0 \vee h_1)\}\end{array}}$$

$$\frac{\{\varphi\}\ P\ \{\psi_1, h_1\} \quad \{\varphi\}\ Q\ \{\psi_2, h_2\}}{\{\varphi\}\ P \| Q\ \{\psi_1 \wedge \psi_2, ((h_1 \widehat{\ } \text{True}) \wedge h_2) \vee (h_1 \wedge (h_2 \widehat{\ } \text{True}))\}}$$

$$\frac{\{\varphi\}\ P\ \{\psi_1, h_1\} \quad \{\psi_1\}\ Q\ \{\psi_2, h_2\}}{\{\varphi\}\ P; Q\ \{\psi_2, h_1 \widehat{\ } h_2\}} \qquad \frac{\{\varphi \wedge \omega\}\ P\ \{\psi_1, h_1\}}{\{\varphi\}\ \omega \rightarrow P\ \{(\varphi \wedge \neg\omega) \vee \psi_1, \ell = 0 \vee h_1\}}$$

$$\frac{\{\varphi\}\ P\ \{\varphi, Inv\} \quad Inv \widehat{\ } Inv \Rightarrow Inv}{\{\varphi\}\ P^*\ \{\varphi, Inv \vee \ell = 0\}}$$

**Table 3.** An inference system for processes

and variables might be changed correspondingly). For $\&_q(b_1, \cdots, b_n)$, only the quantified $\varphi$ over variables in $mv(b)$ is guaranteed to hold almost everywhere throughout the waiting interval, since some binders $b_i$s that make $q$ true might occur at sometime during the interval and as a consequence variables in $\varphi$ might get changed.

For continuous evolution, the notion of differential invariants is used instead of explicit solutions. A *differential invariant* of $\langle \mathcal{F}(\dot{s}, s) = 0 \& B\rangle$ for given initial values of $s$ is a first-order formula of $s$, which is satisfied by the initial values and also by all the values reachable by the trajectory of $s$ defined by $\mathcal{F}$ within the domain $B$. A method on generating differential invariants for polynomial differential equations was proposed in [9]. Here we assume $Inv$ is a differential invariant with respect to precondition $\varphi$ for the continuous evolution (more details on using $Inv$ are shown in the later example proof). For the postcondition, the quantified $\varphi$ over the only modified variables $s$, the closure of $\neg B$, and the closure of $Inv$ hold. The closure $cl(\cdot)$ extends the domain defined by the corresponding formula to include the boundary. For the history formula, the execution interval may be 0, or otherwise, the quantified $\varphi$ over $s$, $B$ and $Inv$ holds almost everywhere throughout the interval.

For communication interrupt, if $b$ fails to occur before the continuous evolution terminates, the effect of the whole statement is almost equivalent to the continuous evolution, except that some variables in $b$ may get changed because of occurrences of some communications during the execution of the continuous evolution. Otherwise, if $b$ succeeds within the termination of the continuous evolution, the continuous evolution will be interrupted and $Q$ will start to execute from the interrupting point. At the interrupting point, the acknowledgement of $b$ holds, and moreover, because $s$ and variables in $mv(b)$ may have been modified, $\exists mv(b).((\exists s.\varphi) \wedge cl(Inv))$ holds (the closure here is

to include the case when the interrupting point is exactly the termination point of the continuous evolution). For the second case, the postcondition is defined as the one for $Q$, and the history formula as the chop of the one for the continuous evolution before interruption and the one for $Q$ afterwards. Finally, as indicated by the rule, the postcondition and history formula for the whole statement are defined as the disjunction of the above two cases.

The rule for $P \| Q$ is defined by conjunction, however, because $P$ and $Q$ may terminate at different time, the formula True is added to the end of the history formula with short time interval to make the two intervals equal. For $P; Q$, the history formula is defined by the concatenation of the ones of $P$ and $Q$. The rule for $\omega \rightarrow P$ includes two cases depending on whether $\omega$ holds or not. At last, for $P^*$, we need to find the invariants, i.e. $\varphi$ and $Inv$, for both the postcondition and history formula.

The general inference rules that are applicable to all constructs, like monotonicity, case analysis etc., can be defined as usual and are omitted here.

We have proved the following soundness theorem:

**Theorem 1.** *Given a process $P$, if $\{\varphi\}\ P\ \{\psi, HF\}$ can be deduced from the inference rules, then $\models \{\varphi\}\ P\ \{\psi, HF\}$.*

PROOF.The proof can be found at the report [16]. □

### 4.2 Application: Reachability Analysis

The inference system can be applied directly for reachability analysis. Given a labelled process $S$ (a process annotated with integers denoting locations), a precondition $\varphi$ and a location $l$ in $S$, by applying the inference system, we can deduce a property $\psi$ such that if $S$ reaches $l$, $\psi$ must hold at $l$, denoted by $\vdash S, l, \varphi \blacktriangleright \psi$. In another word, If $\vdash S, l, \varphi \blacktriangleright \psi$ and $\psi$ is not satisfiable, then $l$ will not be reachable in $S$ with respect to $\varphi$. We have the following facts based on the structural induction of $S$:

- for any process $P$, $\vdash {}^{l}P, l, \varphi \blacktriangleright \varphi$ and $\vdash P^{l}, l, \varphi \blacktriangleright \psi$ provided $\{\varphi\}\ P\ \{\psi, -\}$;
- $\vdash \langle \mathcal{F}(\dot{s}, s) = 0 \& B\rangle \trianglerighteq {}^{l}b \rightarrow S', l, \varphi \blacktriangleright \varphi$. $\vdash \langle \mathcal{F}(\dot{s}, s) = 0 \& B\rangle \trianglerighteq b^{l} \rightarrow S', l, \varphi \blacktriangleright (\exists mv(b).(\exists s.\varphi) \wedge cl(Inv)) \wedge \alpha$ (denoted by $\varphi'$), if $\vdash b \blacktriangleright \alpha$ holds. $\vdash \langle \mathcal{F}(\dot{s}, s) = 0 \& B\rangle \trianglerighteq b \rightarrow S', l, \varphi \blacktriangleright \psi$ if $l \in S'$ and $\vdash S', l, \varphi' \blacktriangleright \psi$ hold;
- $\vdash S_1; S_2, l, \varphi \blacktriangleright \psi$ if $l \in S_1$ and $\vdash S_1, l, \varphi \blacktriangleright \psi$ hold. $\vdash S_1; S_2, l, \varphi \blacktriangleright \psi'$ if $l \in S_2$, $\{\varphi\}\ S_1\ \{\psi, -\}$ and $\vdash S_2, l, \psi \blacktriangleright \psi'$ hold;
- $\vdash \omega^{l} \rightarrow S', l, \varphi \blacktriangleright \varphi \wedge \omega$. $\vdash \omega \rightarrow S', l, \varphi \blacktriangleright \psi$ if $l \in S'$ and $\vdash S', l, \varphi \wedge \omega \blacktriangleright \psi$;
- $\vdash S'^{*}, l, \varphi \blacktriangleright \psi$, if $l \in S'$, $\vdash S', l, \varphi \blacktriangleright \psi$ and $\{\varphi\}\ S'\ \{\varphi, -\}$ hold.

Obviously, the monotonicity holds: if $\vdash S, l, \varphi \blacktriangleright \psi$ and $\psi \Rightarrow \psi'$, then $\vdash S, l, \varphi \blacktriangleright \psi'$.

*Example 6.* Consider $P_0$ in Example 2. Given precondition $\varphi$ , we have $\vdash P_0, 1, \varphi \blacktriangleright (\exists t.\varphi) \wedge t = 0$, denoted by $\varphi_1$. Moreover, $\vdash P_0, 5, \varphi \blacktriangleright (\exists mv(b_0).(\exists s, v, t.\varphi_1) \wedge t \leq T) \wedge (u_a = 1 \vee w_a = 1) \wedge (u_a = 0 \wedge w_a = 0)$, the formula is un-satisfiable, thus location 5 is not reachable. Other locations can be considered similarly. □

$$
\begin{aligned}
\mathsf{TR} = {} & \mathsf{MV}(t_1, T_1) \trianglerighteq {}^{0}\&_{\exists}(\mathsf{trd!}v\{u_v\}, \mathsf{trv!}v\{w_v\})^{7} \\
& \rightarrow (u_v = 1 \wedge w_v = 1 \rightarrow (\mathsf{MV}(t_2, T_2) \trianglerighteq \&_{\exists}(\mathsf{dr?}x_a\{u_a\}, \mathsf{vc?}y_a\{w_a\}) \rightarrow \\
& \quad (w_a = 1 \rightarrow (VA(v, y_a) \rightarrow a := y_a; \neg VA(v, y_a) \rightarrow \mathsf{SC}); \\
& \quad u_a = 1 \wedge w_a = 0 \rightarrow (VA(v, x_a) \rightarrow a := x_a; \neg VA(v, x_a) \rightarrow \mathsf{SC}); \\
& \quad u_a = 0 \wedge w_a = 0 \rightarrow {}^{2}\mathsf{skip});\ t_2 \geq T_2 \rightarrow \mathsf{SC}; \\
& u_v = 1 \wedge w_v = 0 \rightarrow (\mathsf{MV}(t_2, T_2) \trianglerighteq \&_{\exists}(\mathsf{dr?}x_a\{u_a\}) \rightarrow \\
& \quad (u_a = 1 \rightarrow (VA(v, x_a) \rightarrow a := x_a; \neg VA(v, x_a) \rightarrow \mathsf{SC}); \\
& \quad u_a = 0 \rightarrow {}^{3}\mathsf{skip}); t_2 \geq T_2 \rightarrow \mathsf{SC}; \\
& u_v = 0 \wedge w_v = 1 \rightarrow (\mathsf{MV}(t_2, T_2) \trianglerighteq \&_{\exists}(\mathsf{vc?}y_a\{w_a\}) \rightarrow \\
& \quad (w_a = 1 \rightarrow (VA(v, y_a) \rightarrow a := y_a; \neg VA(v, y_a) \rightarrow \mathsf{SC}); \\
& \quad w_a = 0 \rightarrow {}^{4}\mathsf{skip}); t_2 \geq T_2 \rightarrow \mathsf{SC}; \\
& u_v = 0 \wedge w_v = 0 \rightarrow {}^{1}\mathsf{skip}); t_1 \geq T_1 \rightarrow \mathsf{SC}; \\
\mathsf{MV\ (t,\ T)} = {} & t := 0; \langle \dot{s} = v, \dot{v} = a, \dot{t} = 1 \& t < T \rangle \\
\mathsf{SC} = {} & a := -c; \langle \dot{s} = v, \dot{v} = a \& v > 0 \rangle; a := 0
\end{aligned}
$$

**Table 4.** The model of **train**

***Implementation*** We have mechanized the whole framework in Isabelle/HOL and implemented an interactive theorem prover for reasoning about hybrid systems modeled using HCSP with binders [3].

## 5 Train Control Example

We apply our approach to the train control system depicted in Fig. 1: firstly, we construct the formal model for the whole system, especially the train; secondly, prove for the train that it is safe against denial-of-service security attack with respect to properties (F1) and (F2); finally, explore the constraints that relate the constants of different components and learn more precise behavior of the train. Assume for the train that its acceleration ranges over $[-c, c]$ for some $c > 0$, and the maximum speed is $v_{max}$.

***Models*** The model of the train is given in Table 4. There are two auxiliary processes: given a clock variable $t$ and time $T$, MV (t, T) defines that the train moves with velocity $v$ and acceleration $a$ for up to $T$ time units; and SC defines the feedback control of the train when the services from the driver or the VC fail: it performs an emergency brake by setting $a$ to be $-c$ , and as soon as $v$ is reduced to 0, resets $a$ to be 0, thus the train keeps still finally. The main process TR models the movement of a train. The train first moves for at most $T_1$ time units, during which it is always ready to send $v$ to the driver as well as the VC along trd and trv respectively. If neither of them responses within $T_1$, indicated by $t_1 \geq T_1$, the self control is performed. Otherwise, if at least one communication occurs, the movement is interrupted and a sequence of case analysis is followed to execute.

[3] The prover, plus the models and proofs related to the train control example in next section, can be found at https://github.com/wangslyl/hcspwithbinders.

$\mathsf{DR} = \text{wait } T_3; {}^{5}\&_{\exists}\mathsf{trd}?v_d\{u_v\};\ {}^{8}u_v = 1$
$\quad \to (v_d \geq (v_{max} - cT_1 - cT_2)$
$\quad\quad \to [\!]_{l\in[-c,0)} d_a := l;$
$\quad v_d < (cT_1 + cT_2) \to [\!]_{l\in[0,c]} d_a := l;$
$\quad v_d \in [cT_1 + cT_2, v_{max} - cT_1 - cT_2)$
$\quad\quad \to [\!]_{l\in[-c,c]} d_a := l;$
$\quad \&_{\exists}(\mathsf{dr}!d_a\{u_a\}, \mathsf{tick}?o\{u_c\}) \to$
$\quad\quad {}^{12}(u_a = 1 \wedge u_c = 1 \to \text{skip};$
$\quad\quad\quad u_a = 1 \wedge u_c = 0 \to \mathsf{tick}?o\{u_c\};$
$\quad\quad\quad u_a = 0 \wedge u_c = 1 \to \text{skip};$
$\quad\quad\quad u_a = 0 \wedge u_c = 0 \to \text{skip})$
$\quad\quad \|\mathsf{CK});$
$\quad u_v = 0 \to \text{skip}$
$\mathsf{CK} = \text{wait } T_5; \mathsf{tick}!\checkmark$

$\mathsf{VC} = \text{wait } T_4; {}^{6}\&_{\exists}\mathsf{trv}?v_r\{w_v\}; {}^{9}w_v = 1$
$\quad \to (v_r \geq (v_{max} - cT_1 - cT_2)$
$\quad\quad \to r_a := -c;$
$\quad v_r < (cT_1 + cT_2) \to r_a := c;$
$\quad v_r \in [cT_1 + cT_2, v_{max} - cT_1 - cT_2)$
$\quad\quad \to [\!]_{l\in[-c,c]} r_a := l;$
$\quad \&_{\exists}(\mathsf{vc}!r_a\{w_a\}, \mathsf{tick}?o\{w_c\}) \to$
$\quad\quad (w_a = 1 \wedge w_c = 1 \to \text{skip};$
$\quad\quad\quad w_a = 1 \wedge w_c = 0 \to \mathsf{tick}?o\{w_c\};$
$\quad\quad\quad w_a = 0 \wedge w_c = 1 \to \text{skip};$
$\quad\quad\quad w_a = 0 \wedge w_c = 0 \to \text{skip})$
$\quad\quad \|\mathsf{CK});$
$\quad w_v = 0 \to \text{skip}$

**Table 5.** The models of **driver** and **VC**

The first case, indicated by $u_v = 1$ and $w_v = 1$, represents that the driver as well as the VC succeed to receive $v$. The train will wait for at most $T_2$ time units for the new acceleration from the driver or the VC along dr and vc respectively, and during the waiting time, it continues to move with the original acceleration. The new acceleration is expected to satisfy a safety condition $VA(v, a)$:

$$(v > v_{max} - cT_1 - cT_2 \Rightarrow -c \leq a < 0) \wedge (v < cT_1 + cT_2 \Rightarrow c \geq a \geq 0)$$
$$\wedge (cT_1 + cT_2 \leq v \leq v_{max} - cT_1 - cT_2) \Rightarrow (-c \leq a \leq c)$$

which implies the boundaries for setting $a$ to be positive or negative and is necessary for keeping the velocity always in $[0, v_{max}]$, otherwise, it will be rejected by the train. If both the driver and the VC fail to response within $T_2$, indicated by $t_2 \geq T_2$, the self control is performed. Otherwise, the following case analysis is taken: If the train receives a value (i.e. $y_a$) from VC, indicated by $w_a = 1$, then sets $y_a$ to be the acceleration if it satisfies $VA$, otherwise, performs self control; if the train receives a value (i.e. $x_a$) from the driver but not from the VC, updates the acceleration similarly as above; if the train receives no value from both (in fact never reachable), the skip is performed.

The other three cases, indicated by $u_v = 1 \wedge w_v = 0$, $u_v = 0 \wedge w_v = 1$, and $u_v = 0 \wedge w_v = 0$, can be considered similarly.

One possible implementation for driver and VC is given in Table 5, in which process wait $T_i$ for $i = 3, 4$ is an abbreviation for $t_i := 0; \langle \dot{t}_i = 1 \& t_i < T_i \rangle$. In process DR, the driver asks the velocity of the train every $T_3$ time units, and as soon as it receives $v_d$, indicated by $u_v = 1$, it computes the new acceleration as follows: if $v_d$ is almost reaching $v_{max}$ (by the offset $cT_1 + cT_2$), then chooses a negative in $[-c, 0)$ randomly; if $v_d$ is almost reaching 0, then chooses a non-negative in $[0, c]$ randomly; otherwise, chooses one in $[-c, c]$ randomly. The train then sends the value being chosen (i.e. $d_a$) to the train, and if it fails to reach the train within $T_5$ (i.e. the period of the clock), it will give up. The auxiliary process clock is introduced to prevent deadlock caused by the situation when the driver succeeds to receive velocity $v_d$ from the train but fails to send acceleration $d_a$ to the train within a reasonable time (i.e. $T_5$ here). VC and DR

have very similar structure, except that $\mathsf{VC}$ has a different period $T_4$, and it will choose $-c$ or $c$ as the acceleration for the first two critical cases mentioned above.

Finally, the train control system can be modeled as the parallel composition: $\mathsf{SYS} = \mathsf{TR}^* \| \mathsf{DR}^* \| \mathsf{VC}^* \| \mathsf{CK}^*$. By using $^*$, each component will be executed repeatedly.

***Proofs of Train*** First of all, we define the precondition of $\mathsf{TR}^*$, denoted by $\varphi_0$, to be $VA(v,a) \wedge 0 \leq v \leq v_{max} \wedge -c \leq a \leq c$, which indicates that in the initial state, $v$ and $a$ satisfy the safety condition and are both well-defined.

Secondly, we need to calculate the differential invariants for differential equations occurring in $\mathsf{TR}$. Consider the equation in $\mathsf{MV}(t_1, T_1)$, the precondition of it with respect to $\varphi_0$, denoted by $\varphi_1$, can be simply calculated, which is $\varphi_0 \wedge t_1 = 0$, then by applying the method proposed in [9]:

$$\begin{pmatrix} \big(0 \leq t_1 \leq T_1\big) \bigwedge \\ \big(a < 0 \Rightarrow (v \geq cT_2 + (at_1 + cT_1)) \wedge (v \leq v_{max})\big) \\ \bigwedge \big(a \geq 0 \Rightarrow (v \leq v_{max} - cT_2 + (at_1 - cT_1)) \wedge (v \geq 0)\big) \end{pmatrix}$$

denoted by $Inv_1$, constitutes a differential invariant of the continuous with respect to $\varphi_1$. It is a conjunction of three parts, indicating that: (1) $t_1$ is always in the range $[0, T_1]$; (2) if $a$ is negative, $v$ must be greater or equal than $cT_2$ plus a positive value (i.e. $at_1 + cT_1$), and meanwhile $v \leq v_{max}$; and (3) if $a$ is positive, $v$ must be less or equal than $v_{max} - cT_2$ plus a negative value (i.e. $at_1 - cT_1$), and meanwhile $v \geq 0$. This invariant is strong enough for guaranteeing $cT_2 \leq v \leq v_{max} - cT_2$ after the continuous escapes no matter what $a$ is in $[-c, c]$. Similarly, we can calculate the invariant of the continuous occurring in $\mathsf{MV}(t_2, T_2)$, which is

$$\begin{pmatrix} \big(0 \leq t_2 \leq T_2\big) \bigwedge \\ \big(a < 0 \Rightarrow (v \geq 0 + (at_2 + cT_2)) \wedge (v \leq v_{max})\big) \\ \bigwedge \big(a \geq 0 \Rightarrow (v \leq v_{max} + (at_2 - cT_2)) \wedge (v \geq 0)\big) \end{pmatrix}$$

denoted by $Inv_2$. This invariant is strong enough for guaranteeing $0 \leq v \leq v_{max}$ after the continuous escapes. Finally, the invariant of the differential equation of $\mathsf{SC}$ is $0 \leq v \leq v_{max}$, and we denote it by $Inv_3$.

Next, to prove (F1) and (F2), we can prove the following facts instead:

- Locations 1, 2, 3, 4 are not reachable for $\mathsf{TR}^*$;
- Throughout the execution of $\mathsf{TR}^*$, the invariant $0 \leq v \leq v_{max}$ always holds.

First we consider one loop of execution $\mathsf{TR}$. For location 1, we can deduce that [4] $\vdash \mathsf{TR}, 1, \varphi_0 \blacktriangleright (\mathbf{u_v} \vee \mathbf{w_v}) \wedge (\neg \mathbf{u_v} \wedge \neg \mathbf{w_v})$, which is not satisfiable, thus location 1 is never reachable. Similarly, we can deduce that locations 2, 3, 4 are not reachable as well. Furthermore, by applying the inference system, we can deduce the specification $\{\varphi_0\}\ \mathsf{TR}\ \{\varphi_0, \lceil 0 \leq v \leq v_{max} \rceil^<\}$. After one loop of execution of the train, $\varphi_0$ still holds at termination. Thus, all the above reachability results obtained for $\mathsf{TR}$ still hold for $\mathsf{TR}^*$, whose execution is equivalent to some finite number of executions of

[4] For simplicity, we use the boldface of an acknowledgment variable to represent the corresponding formula, e.g., $\mathbf{u_v}$ for $u_v = 1$.

$\mathsf{TR}$. Finally, plus that $\lceil 0 \leq v \leq v_{max} \rceil^{<}$ is idempotent over chop, we can deduce $\{\varphi_0\}\ \mathsf{TR}^*\ \{\varphi_0, \lceil 0 \leq v \leq v_{max} \rceil^{<}\}$, denoted by (**TrainSpec**), which implies that $0 \leq v \leq v_{max}$ is an invariant for the train.

By applying our interactive theorem prover, the fact (**TrainSpec**) is proved as a theorem, and the above reachability results can be implied from the lemmas proved for corresponding processes, according to the method introduced in Section 4.2.

We can see that, most of the proofs need to be performed in an interactive way, mainly because of the following reasons: firstly, we need to provide the differential invariants by ourselves during proof of continuous evolution; and secondly, we need to conduct the proof of DC formulas by telling which axiom or inference rule of DC should be applied. For the first problem, we will consider the integration of the prover to a differential invariant generator that can be implemented based on the method proposed in [9]. For the second, we will consider the decidability of DC and design algorithms for solving the decidable subsets, or as an alternative approach, consider translating DC formulas into HOL formulas in a semantic way and applying the existing automatic solvers for HOL instead. Both of these will be our future work.

***Constraints of Constants*** We can further analyze the behavior of the whole system $\mathsf{SYS}$. By defining the constraints relating different constants, the behavior of communications between the three components can be determined. Consider the first loop of execution of each component, based on reachability analysis, we have the following facts: for locations 0, 5, 6, $t_1 = 0$, $t_3 = T_3$ and $t_4 = T_4$ hold respectively, and for locations 7, 8, 9, $t_1 \leq T_1$, $t_3 \geq T_3$ and $t_4 \geq T_4$ hold respectively. The synchronization points have four possibilities: $(7, 8)$, $(7, 9)$, $(7, 8, 9)$, or none. For the first case, i.e. the train succeeds to communicate with the driver but not with the VC, there must be $t_1 = t_3 < t_4$, and if $T_3 < T_4$ and $T_3 \leq T_1$ hold, this case will occur. The second one is exactly the contrary case. For the third case, there must be $t_1 = t_3 = t_4$, and if $T_3 = T_4 \leq T_1$ holds, this case will occur. Finally, if both $T_3 > T_1$ and $T_4 > T_1$ hold, the last case occurs, i.e., locations 7, 8 and 9 are not reachable, and thus the train fails to communicate with both the driver and the VC. Following this approach, more precise behavior of the communications of the train can be obtained.

## 6 Conclusion and Future Work

This paper proposes a formal modeling language, that is a combination of hybrid CSP and binders from quality calculus, for expressing denial-of-service due to unreliable communications in hybrid systems. With the linguistic support, it is able to build a safe hybrid system that behaves in a reasonable manner in the presence of denial-of-service security attack. The paper also develops an inference system for reasoning about such systems, with no dependence on the behavior of the environment, and furthermore implements an interactive theorem prover. We illustrate our approach by considering an example taken from train control system.

The investigation of our approach to more complex hybrid systems is one of our future work. Meanwhile, for facilitating practical applications, we will consider to achieve more support of automated reasoning in the theorem prover.

**Acknowledgements.** The research has been supported by NSFC-6110006 and NSFC-91118007, supported by the National Natural Science Foundation of China, and by IDEA4CPS, supported by the Danish Foundation for Basic Research.

## 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 736*, pages 209–229, 1992.
2. R. Alur, T. Dang, and F. Ivancic. Predicate abstraction for reachability analysis of hybrid systems. *ACM Trasactions on Embedded Computing Systems*, 5(1):152–199, 2006.
3. E. Asarin, O. Bournez, T. Dang, and O. Maler. Approximate reachability analysis of piecewise-linear dynamical systems. In *HSCC'00, LNCS 1790*, pages 21–31, 2000.
4. E. M. Clarke, A. Fehnker, Z. Han, B. H. Krogh, J. Ouaknine, O.Stursberg, and M. Theobald. Abstraction and counterexample-guided refinement in model checking of hybrid systems. *nt. J. Found. Comput. Sci.*, 14(4):583–604, 2003.
5. J. He. From CSP to hybrid systems. In *A classical mind*, pages 171–189. Prentice Hall International (UK) Ltd., 1994.
6. T. A. Henzinger. The theory of hybrid automata. In *LICS'96*, pages 278–292, 1996.
7. G. Lafferrierre, G. J. Pappas, and S. Yovine. Symbolic reachability computation for families of linear vector fields. *Journal of Symbolic Computation*, 11:1–23, 2001.
8. J. Liu, J. Lv, Z. Quan, N. Zhan, H. Zhao, C. Zhou, and L. Zou. A calculus for hybrid CSP. In *APLAS'10, LNCS 6461*, pages 1–15. 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. N. Lynch, R. Segala, F. Vaandrager, and H. Weinberg. Hybrid I/O automata. In *HSCC'96, LNCS 1066*, pages 496–510, 1996.
11. Z. Manna and A. Pnueli. Verifying hybrid systems. In *Hybrid Systems, LNCS 736*, pages 4–35. Springer, 1993.
12. H. Riis Nielson and F. Nielson. Probabilistic analysis of the quality calculus. In *FORTE'13, LNCS 7892*, pages 258–272. Springer, 2013.
13. H. Riis Nielson, F. Nielson, and R. Vigo. A calculus for quality. In *FACS'13, LNCS 7684*, pages 188–204. Springer, 2013.
14. A. Platzer. Differential-algebraic dynamic logic for differential-algebraic programs. *J. Log. and Comput.*, 20(1):309–352, 2010.
15. A. Platzer and J. Quesel. European Train Control System: A case study in formal verification. In *ICFEM'09, LNCS 5885*, pages 246–265. Springer, 2009.
16. S. Wang, F. Nielson, and H. Riis Nielson. A framework for hybrid systems with denial-of-service security attack. Technical Report ISCAS-SKLCS-14-06, Institute of Software, Chinese Academy of Sciences, 2014.
17. N. Zhan, S. Wang, and H. Zhao. Formal modelling, analysis and verification of hybrid systems. In *ICTAC Training School on Software Engineering, LNCS 8050*, pages 207–281, 2013.
18. C. Zhou and M.R. Hansen. *Duration Calculus — A Formal Approach to Real-Time Systems*. Monographs in Theoretical Computer Science. An EATCS Series. Springer-Verlag Berlin Heidelberg, 2004.
19. C. Zhou, C.A.R. Hoare, and A. P. Ravn. A calculus of durations. *Information Processing Letters*, 40(5):269–276, 1991.
20. C. Zhou, J. Wang, and A. P. Ravn. A formal description of hybrid systems. In *Hybrid systems, LNCS 1066*, pages 511–530. Springer, 1996.